Selective Querying For Adapting Web Service Compositions Using the Value of Changed Information

by

John Harney

(Under the direction of Prashant Doshi)

Abstract

Web services are autonomous, reusable, platform-independent software applications that can be accessed over the Web. One of the key advantages of utilizing Web services is that they may be quickly composed to form larger processes of varying complexity to provide functionality that none of the individual component services could provide alone. Consequently, organizations have increasingly adopted the use of Web service compositions to assemble complex, interoperable applications in a variety of domains. The decentralized nature of compositions imply that they typically function in volatile environments where the parameters of the participating Web services change during execution. Thus, it is critical to adapt to these changes promptly and properly, so as to maintain a consistent and optimal composition. One traditional approach for adapting WSCs dynamically is to identify and query for the vital changes in the data of the various component services, and then integrate the revised data into the composition model. Queries are often costly and time-consuming, however, and must be carefully managed. In this thesis, I present a method that selectively queries component Web services for their revised values by using the value of changed information (VOC) approach. VOC measures the value of the change that revised information may potentially introduce to the composition. This metric is used to determine whether or

not a query should be issued to a component service for its revised value. In doing so, we may focus only on those queries that obtain information that will greatly impact the composition and eliminate costly queries that are not needed. As computing VOC comes at a computational price, I present techniques for alleviating its complexity so that it becomes a more feasible approach. Furthermore, I present approaches that generalize the VOC approach so that it may be used in a wide array of composition configurations and accommodate compositions that are sensitive to risks. In all, I demonstrate empirically and theoretically that the VOC-driven selective querying method compares favorably to other state-of-the-art querying approaches for adaptation.

INDEX WORDS: Value of Information, Modeling, Services-Oriented Computing, Web Services, Web Services Compositions, Adaptation, Theses (academic)

Selective Querying For Adapting Web Service Compositions Using the Value of Changed Information

by

John Harney

B.S., North Carolina State University, 2000

M.S., The University of North Carolina-Greensboro, 2005

A Dissertation Submitted to the Graduate Faculty

of The University of Georgia in Partial Fulfillment

of the

Requirements for the Degree

Doctor of Philosophy

Athens, Georgia

2010

Selective Querying For Adapting Web Service Compositions Using the Value of Changed Information

by

John Harney

Approved:

Major Professor: Prashant Doshi

Committee: Krzysztof J. Kochut
John Miller
Biplav Srivastava

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
August 2010

Acknowledgments

I would be remiss if I did not thank all (or at least some) of the people that have aided me in my journey through the rigors of creating this thesis. First and foremost, I would like to acknowledge the contributions of my advisor, Dr. Prashant Doshi. He is unquestionably a true innovator in the field of computer science and a great teacher and motivator. I feel that he has prepared me for success as a computer scientist and I have learned a great deal from him. Through his guidance and motivation, my graduate study here at the University of Georgia has been an enjoyable learning experience. Second, I would like to thank my committee members, Dr. Krzysztof Kochut and Dr. John Miller, for their input, valuable discussions and accessibility. I would especially like to thank my fourth committee member, Dr. Biplav Srivastava, who, in addition to collaborating with us on past projects, has helped shape my career path by providing a key reference in my job search. In a similar vein, I would like to thank Dr. David Lowenthal, for his helpful career advice and heated sports debates. Third, I would like to thank Dr. Bob Bostrom and Dr. Jaxk Reeves, who provided critical counsel on topics required in this thesis that were outside of my understanding. Fourth, I would like to thank my labmates and closest friends I have come to meet over the past few years (and in my lifetime) for helping me keep my sanity (at times) and perspective on what is truly important in life – unfortunately if I took the time to name them all I would double the length of this document! Among them I should mention Haibo Zhao, who has fought with me in the trenches since day one of this long journey. Finally, I would like to thank the fam – Mom, Dad, Missy, Andy, and of course Punky – for their unconditional love and support throughout my whole life. I am truly blessed to have such a loving family by my side when I have nowhere else to turn.

This dissertation work was partially funded by the Computer Science Department of the University of Georgia.

Preface

My dissertation research focuses on addressing the problem of adapting Web service compositions (WSCs) that function in environments where the non-functional, or quality of service (QoS), properties of the individual component services may change during execution. Adapting to these types of changes efficiently is important so as to maintain an optimal composition without incurring much overhead. While many adaptation techniques have been proposed, I focus on the adaptation of WSCs through the use of selectively querying the component services' providers for their revised non-functional data. My research aims in deriving efficient and intelligent ways to selectively query those services whose changes will impact the composition.

Together with my advisor, Dr. Prashant Doshi, I have attempted to disseminate the research work through appropriate mediums such as workshops, conferences, and journal submissions. The list of papers given below along with this dissertation proposal forms an accurate description of the work that I have completed towards my dissertation.

Publication List

1. John Harney and Prashant Doshi, Risk Sensitive Value of Changed Information for Selective Querying of Web Services, *International Conference on Services Oriented Computing (ICSOC 2010)*, in submission.

2. John Harney and Prashant Doshi, Selective Querying for Adapting Hierarchical Web Service Compositions, to appear in*Methodologies for Non-Functional Requirements in Service Oriented Architecture: Requirements Engineering, Model-Driven Development and Security*, to be published by IGI Global in 2010.

3. John Harney and Prashant Doshi, Selective Querying for Adapting Hierarchical Web Service Compositions Using Aggregate Volatility, in Research track in *IEEE International Conference on Web Services (ICWS)* (acceptance rate 15.6%) , pages 43–50, 2009.

4. John Harney and Prashant Doshi, Selective Querying for Adapting Web Service Compositions Using the Value of Changed Information, in *IEEE Transactions on Services Computing (TSC)*, Vol. 1, No. 3, pages 169–185, 2008.

5. John Harney and Prashant Doshi, Speeding Up Web Service Composition with Volatile External Information, in *International Workshop on Context Enabled Source and Service Selection, Integration, and Adaptation (CSSSIA)*, World Wide Web, Vol 294, pages 1–7, 2008.

6. John Harney and Prashant Doshi, Speeding Up Web Service Composition with Volatile External Information, in *the World Wide Web Conference (WWW)*, pages 1201–1202, 2008.

7. Girish Chafle, Prashant Doshi, John Harney, Sumit Mittal and Biplav Srivastava, Improved Adaptation of Web Service Compositions Using Value of Changed Information, in Applications and Industry track, in *IEEE International Conference on Web Services (ICWS)* (acceptance rate 18%), pages 784–791, 2007.

8. John Harney and Prashant Doshi, Speeding Up Adaptation of Web Service Compositions Using Expiration Times, in Research track in *the World Wide Web Conference (WWW)* (acceptance rate 14.8%) , pages 1023–1032, 2007.

9. John Harney and Prashant Doshi, Adaptive Web Process Composition Using Value of Changed Information, in Research track in *International Conference on Service Oriented Computing (ICSOC)* (acceptance rate 15%), pages 784-791, 2006

10. John Harney and Prashant Doshi, Adaptive Web Processes Using Value of Change Computations, in *Workshop on AI-Driven Technologies for Services-Oriented Computing*, AAAI, pages 19–25, 2006.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

To satisfy the growing demand for distributed applications, service-oriented architectures (SOAs) have been adopted as a viable solution for enterprise applications. SOA is a software engineering paradigm for organizing and utilizing distributed Web services (WSs) – autonomous, reusable, and platform-independent entities that can be accessed via the Web. One of the key benefits of utilizing services is the potential for automatically formulating compositions of services resulting in integrated software applications. Consequently, as organizations are increasingly adopting the use of Web services to quickly assemble interoperable applications, Web service compositions (WSCs) are providing solutions for a wide array of problem domains.

WSCs seek to find an optimal composition of available Web services to satisfy a given business goal. Many of the proposed approaches for optimally composing Web services predominantly utilize fixed models designed prior to runtime [85, 54, 3, 2, 63]. These approaches often assume that the parameters of the participating WSs that are used to model the environment remain static and accurate throughout its execution. The compositions are built using a pre-defined model of the environment obtained at design time and subsequently executed with the assumption that the environment will not change. This fundamental assumption is unrealistic, as WSCs are often executed in *volatile* environments, where the parameters of the participating WSs are subject to change during execution. External events, whether anticipated or unforeseen, may occur during execution, consequently triggering changes in the component WS's parameters. For example, a product may go out of stock affecting its availability, the network bandwidth may fluctuate affecting the WS response time, or costs

of using a travel agent's service may increase. When ignored, these types of changes may negatively impact the performance of the WSC. It is therefore critical that the compositions identify and adapt to the changes appropriately, so as to avoid undesirable consequences, such as an unwanted consumption of time, money, or other valuable resources.

Studies [68, 61] have shown that volatility can manifest in WSC environments in a variety of ways. Changes range from the operational level (such as a newly introduced workflow task) to the organizational level (such as new company policies). Solutions have been presented to address some these changes, ranging from exception-handling techniques defined in [13] to instituting protocol adaptations defined in [25].

Less attention, however, has been paid to the *data volatility* that exists during execution of compositions. As a concrete example, consider a manufacturer who wishes to implement its supply chain (shown in Figure 1.1) as a WSC. It is a small portion of a parts procurement and assembly process, where the manufacturer must obtain a specific part to finish assembling a larger product for delivery. The manufacturer has an option between different WS providers from whom to obtain specific parts. The first option is to obtain the parts from its Inventory WS, an inexpensive option that would allow the manufacturer to acquire the part quickly and cheaply. The manufacturer has limited storage available, however, making this method of obtaining the part unreliable. The manufacturer may also choose to obtain the parts directly from a Preferred Supplier (or Other Suppliers with whom the manufacturer has previously interacted).The part is more expensive to obtain from these suppliers, but its availability is significantly better than the Inventory. Finally, the manufacturer may rely on the Spot Market, which almost certainly guarantees that the part will be obtained, but is more expensive than the previous options.

The ordering of the manufacturer's actions depends on both the probability with which a supplier usually satisfies the orders and the cost of using a supplier's WS. In Figure 1.1, one might say that the Preferred Supplier would be the best option, as it balances risk (its rate of order satisfaction is higher than Inventory) with reward (its cost is lower than the Spot

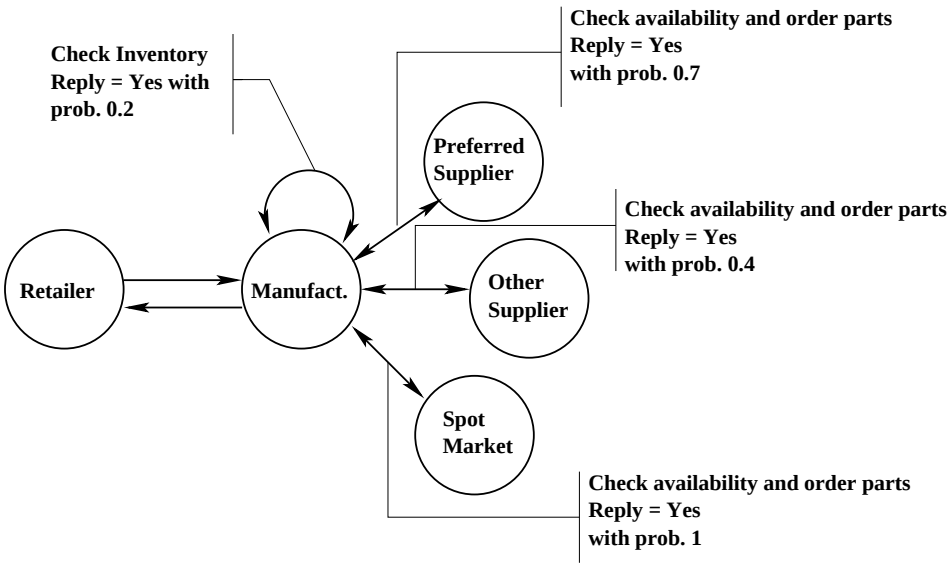


Figure 1.1: Collaboration diagram showing interactions between the business partners. The manufacturer may choose between obtaining a part from its own Inventory, a Preferred Supplier service, another supplier (Other Supplier) or the rely on the Spot Market. The Inventory supplies parts cheaper but is less available, the Spot Market can provide parts reliably but expensively and the Preferred Supplier has moderate cost and availability of parts. Example probability values have been included to aid understanding.

Market). However, if the Preferred Supplier's rate of order satisfaction drops suddenly (due to say, a warehouse fire), a cost-conscious manufacturer should replace it with another supplier to remain optimal. If the manufacturer does not revise its composition model to reflect this change, however, it will continue to utilize the Preferred Supplier over other candidate WSs, potentially incurring an otherwise avoidable cost.

Important service parameters such as cost, availability, or the rate of order satisfaction in the above example, often change during the life-cycle of a WSC. WSCs must be aware of the changing parameters of the participating services so as to optimize the composition. Thus, the WSC must possess up-to-date knowledge of the revised information during execution. To obtain this knowledge, an adaptive WSC may query component services – typically their

providers – for their revised parameter values. The changed values can be then integrated into the composition model so that it may remain optimal.

1.1 Problem Definition

Recently, several approaches [71, 20, 9] have been proposed to combat data volatility through the creation of monitoring systems that query service providers for their revised parameters. The procedure of querying for component services' parameters, however, comes with its own attendant challenges that are often overlooked by these frameworks. While revised information on some services may cause changes in the overall WSC, changes to other services' parameters may have little or no impact on the WSC. Additionally, WSCs typically operate over an open and large-scale system (the Internet). As a result, querying for information from service resources is often tedious, time consuming, and costly. Queries must therefore be carefully managed – we should only query those services whose parameter changes may potentially impact the WSC so as to minimize the additional overhead introduced.

Indeed, there are many factors that should be considered when designing approaches for querying revised information in volatile composition environments. Specifically, a proper querying method should:

- **Determine if the revised parameter information of a service is *useful* and *cost-efficient*.** Not all changes in the component WSs will affect the performance of the composition. We should identify only those services whose parameter changes will substantially impact the composition, so that we do not waste critical resources on unnecessary queries. Furthermore, we should only query if the potential cost savings realized by obtaining the new information outweigh the cost of querying.

- **Be *computationally tractable.*** It is critical for querying approaches to be practical to use.

- **Understand how to query in order to adapt *nested* configurations of services in a WSC.** Many real-world WSCs consist of services arranged in various configurations (i.e. sequential, concurrent, branches, etc.). Furthermore, in many cases, a WSC may be seen as nested – a higher level WSC may be composed of WSs and lower level WSCs – which induces a natural hierarchy over the composition. The method of querying can be supported on WSCs containing a wide variety of component service configurations.

- **Allow for considerations of a composition user's *risk preferences*.** Distinct preferences toward risk could significantly affect which WSs are selected in the composition and how the composition is adapted.

The challenges above demonstrate that adapting compositions to changes in the environment through the procedure of querying is a difficult process that must be carefully managed. Thus, producing a sophisticated selective querying technique that manages queries efficiently becomes an important step in realizing a composition capable of adapting to change.

1.2 Contributions

My research is motivated by the issues discussed above. I performed an extensive review of the existing WSC adaptation approaches and specifically, I identified the key issues above that need to be resolved in adapting WSCs in the presence of data volatility. I have proposed a novel selective querying technique that addresses the challenges stated above and demonstrated its performance theoretically and empirically. As a result, the approach has seen significant improvements over existing querying heuristics, and demonstrates cost savings in WSCs that function in volatile environments. In this section, I will summarize the dissertation research that I have accomplished thus far, and give further details in subsequent chapters.

1.2.1 The Value of Changed Information

As discussed previously, the parameters of the participating services may change during the life-cycle of a WSC. To adapt to these changes, I introduce a selective querying scheme that will suggest a query only when the queried information is expected to be sufficiently valuable. This approach draws inspiration from an important concept in the Artificial Intelligence literature [59] called the Value of Perfect Information (VPI), which determines the amount of resources (e.g. time, money, etc) a decision making agent would be willing to pay for information prior to making a decision.

I introduce a new concept for information revision called the *Value of Changed Information* (VOC) [33]. VOC essentially measures how badly, on average, the original composition will perform in a changing environment. VOC is then used to guide our selective querying methodology. Although VOC shares its conceptual underpinnings with VPI, they differ in one very important way – VPI computes the value of *additional* information, while the VOC provides the value of *revised* information.

The VOC-driven querying approach is myopic in that only one service is considered for querying at a time. At each stage of the WSC, we find the service with the greatest VOC value (denoted as VOC^*) and compare that value to the cost of obtaining the service's revised parameter values (referred to as a $QueryCost$). If the VOC^* is found to be greater than the $QueryCost$, then a query is issued to that service, the revised parameter information is integrated into the model, and the WSC is recomposed; the composition then continues execution. If the VOC is less than the $QueryCost$, then no query is issued as the revised information is not expected to be worth the cost of obtaining it, and the execution of the composition continues.

The major contributions of this work are listed below:

Contributions

- Identification of services whose expected parameter changes impact the overall performance of the WSC. Not all changes to parameters of component services upset

the performance of the WSC. Additionally, changes in some WSs may have a greater bearing on the performance of the WSC than others. VOC quantifies the impact that a component WS's revised information is expected to introduce to the WSC, and identifies candidate services for querying.

- Determination of whether queries are worth issuing. We should only query for new data if this new information is worth the price of obtaining it. VOC provides us a well-defined threshold as a way to comparing the value that new information may bring to the WSC and the actual cost of querying.

- Demonstration of experimental results showing that WSC adaptation using the VOC-driven querying strategy outperforms other naive querying heuristics.

These contributions are outlined in Chapter 4.

1.2.2 Mitigating Complexity of VOC

Using the VOC-driven selective querying strategy may be computationally intensive. Computational lag can be detrimental to a composition, as it may consume a large quantity of computing resources as well as add time to the execution of the WSC, which may render the technique impractical to use. I present two techniques [36, 34] that mitigate computational complexity that accompanies finding VOC.

First, I alleviate the complexity of finding the VOC for an individual service. We will observe that for particular values of the parameters, the WSC remains unchanged. Consequently, such revised values may be ignored, as they do not change the WSC. I provide a simple and quick way to ascertain these values.

Second, I reduce the time required to compute the VOC of the candidate service with the largest VOC, or VOC^*. This can be done by utilizing the expiration times often associated by service providers through a service level agreement (SLA) to the parameters of their services. We may use the intuition that we need not consider querying those services

for revised information whose previously obtained information has not expired. I incorporate this insight into the VOC formulation, and call the approach, the value of changed information with expiration times (VOC^{ϵ}). Because VOC^{ϵ} focuses the computations on only those services whose parameters could have changed, it is computationally more efficient than the traditional VOC.

The major contributions of this work are listed below:

Contributions

- Reduction of the inherent computational complexity of finding the VOC of an individual service. This is done through the identification of revised service parameter values that will not impact the WSC. These values, found in constant time, eliminate much of the computational overhead involved.

- Reduction of finding VOC^{*} by introducing VOC^{ϵ}-driven selective querying. VOC^{ϵ} exploits the expiration time information given in pre-defined service-level agreements by identifying those services whose parameters have not expired. Services that have not expired may be eliminated from consideration in computing VOC^{*}, effectively reducing the average run time of the obtaining VOC^{*}.

- Empirical and theoretical demonstration of the reduction in composition time needed for VOC^{ϵ}-driven selective querying in comparison to traditional VOC-driven selective querying.

These contributions are outlined in Chapter 5.

1.2.3 Adapting Hierarchical Web Service Compositions Using VOC

In many cases, a WSC may be seen as nested - a higher level WSC may be composed of WSs and lower level WSCs, which induces a natural hierarchy over the composition. In contrast to flat WSCs, a hierarchical decomposition introduces multiple challenges: (a) Because only the parameters of the lower level component services are known, we must derive

the parameters of the higher-level composite service from these, and, (b) we must derive a model of volatility for composite services' parameters from the models of its component services. While approaches for aggregating QoS parameters of component WSs exist [85, 78], I present a way of formulating the stochastic models of volatility of composite services. Given approaches that address both challenges above, I show how we may adapt hierarchical WSCs by descending down the levels of nesting and computing the VOC for WSCs at each level. The resulting approach performs well in the presence of data volatility, and outperforms outher querying approaches.

The major contributions of this work are listed below:

Contributions

- Derivation of the volatility of the parameters of composite services. The volatility of composite services may be obtained by combining the volatility of its component services' parameters. This allows us to determine if a composite service is a suitable candidate for querying.

- Derivation of the volatility of the parameters of WSs assembled in diference configurations. Analogous to obtaining the volatiliy of a composite service, I derive models of volatility for various workflow configurations (i.e. concurrent, sequential, or branching patterns).

- Demonstration of the runtime complexity using VOC-driven querying in hierarchical Web service compositions

- Demonstration of experimental results showing that adaptation using the VOC-driven querying strategy continues to outperform other naive querying heuristics for any WSC configuration.

These contributions are outlined in Chapter 6.

1.2.4 Risk-sensitive VOC

Approaches such as VOC aim to adapt compositions in a rational - risk neutral - manner. However, risk attitudes often strongly influence the modern organizations decision analysis cycle and play a pivotal role in determining the goals of many different business processes. For example, an organization may be conservative by nature, willing to sacrifice some cost in exchange for more stability and reduced risk of incurring greater costs in the future. In contrast to such risk aversion, risk-seeking behavior involves making decisions that could yield large gains at the risk of incurring heavier losses. Clearly, a comprehensive approach to composition and adaptation should allow considerations of risk preferences. This is because distinct preferences toward risk could significantly affect which WSs are selected in the composition and how the composition is adapted. Motivated by this need, I generalize VOC toward modeling risk preferences in deciding which WSs to query for revised information. Here, I model risk preferences using utility functions [70] a well-known way of modeling different attitudes toward risk. I show that considerations of risk preferences affect which WSs are included for participation in compositions, and which WSs are selected for querying of revised information. This in turn affects how compositions are adapted.

The major contributions of this work are listed below:

Contributions

- Demonstration that considerations of risk preferences affect which WSs are included for participation in compositions. While the rational costs of the compositions may be high, its utility to the risk-sensitive designer is optimal.

- Demonstration of the affect that risk preferences have on adaptation. Risk preferences will affect which WSs is targeted for querying.

These contributions are outlined in Chapter 7.

1.3 Thesis Organization

The rest of this dissertation is outlined as follows. Chapter 2 briefly reviews the general Web service composition problem and introduces the motivation of using a query-based adaptation mechanism in a composition. I also explore the relevant research literature related to the ideas presented here. I describe three motivating scenarios in Chapter 3, which motivate the need for a composition to adapt to a dynamic environment. In Chapter 4, the Value of Changed Information methodology will be introduced. Chapter 5 outlines two methods that reduce complexities in computing VOC. In Chapter 6, I extend VOC for use in larger scale, hierarchical compositions. In Chapter 7, I generalize VOC toward modeling risk preferences in deciding which WSs to query for revised information. Finally, Chapter 8 gives a brief discussion of the accomplished work and outlines some avenues of future work.

Chapter 2

Background and Related Work

In this chapter, I give a brief overview of the ongoing research in Web service compositions, as well as the more recent approaches in resolving the adaptation problem. Much of this previous work forms the necessary foundation from which this thesis is based. Section 2.1 introduces the Web service composition problem and discusses several approaches for composition, including a model that I will follow for the remainder of this thesis. Section 2.2 motivates the need for adaptation in compositions and outlines some of the approaches used to address the adaptation problem in Web service compositions.

2.1 Web Service Compositions

The ability to efficiently and effectively select and integrate Web services is an important step towards the development of highly complex Web applications. To promote such interoperability, several standards have been created. These standards include the Web Service Description Language (WSDL) [75] standard for service description, the Universal Description, Discovery, and Integration (UDDI) [74] standard for service discovery, Simple Object Access Protocol (SOAP) [73] standard for service invocation, and the Business Process Execution Language (BPEL) [14] standard for service orchestration. These standards allow Web services to be combined to form larger processes of varying complexity to provide functionality that none of the individual component services could provide alone. As the demand for quickly assembling services has grown, approaches for automatic composition have been extensively researched.

Although the general problem of automating Web service compositions has many interpretations, its objectives are relatively straightforward – given a formal specification and a set of available Web services, find a service composition which satisfies the specification and optimizes the composition user's benefit. The benefit of a service composition is determined by an aggregation of the quality of service (QoS) properties of its component services. QoS properties include non-functional parameters of the services such as its invocation costs, response time, availability, reliability, etc. These properties are typically defined in contracts between service partners called service-level agreements (SLAs) [1].

I outline a few of the more prevalent composition frameworks next.

2.1.1 Traditional Composition Approaches

Many of the initial approaches for automatic Web service composition employ classical AI planning techniques [55, 63, 54]. Services are represented as actions, and the composition is treated as a planning problem. A "plan" simply becomes a sequence of service invocations that accomplishes a goal or task. These approaches guarantee compositions that meet functional requirements, but are unable to provide optimizations of non-functional (QoS) properties such as cost and availability. Other model-based approaches [3, 2, 81] obtain QoS parameters prior to runtime and compose services based on their parameter values. They apply mathematical optimization techniques such as integer programs to maximize over some QoS criteria defined by the user of a composition.

2.1.2 Composition Using Decision-Theoretic Planning

More recently, decision-theoretic planners were proposed [27] to model compositions and have been applied extensively to composition frameworks [85, 69, 79]. Decision-theoretic planners such as Markov decision processes (MDPs) generalize classical planning techniques to non-deterministic environments, where action (and in this case, WS invocation) outcomes

[1] Standards for SLAs include WS-Policy [77] and WS-Agreement [17]

are uncertain. Decision-theoretic planning formalisms model the uncertainty present in the process and produce a plan that optimally balances risk and reward. These models are especially relevant in the context of domains where processes must minimize costs. Consequently, I adopt this model of composition for illustration for the rest of this thesis [2].

Model

MDPs model the process environment, WP, using a sextuplet:

$$WP = (S, A, T, C, H, s_0)$$

where $S = \Pi_{i=1}^{n} X^i$, where S is the set of all possible states factored into a set, X, of n variables, $X = \{X^1, X^2, \ldots, X^n\}$; A is the set of all possible actions; T is a transition function, $T : S \times A \rightarrow \Delta(S)$, which specifies the probability measure over the next state given the current state and action; C is a cost function, $C : S \times A \rightarrow \mathbb{R}$, which specifies the cost of performing each action from each state; H is the period of consideration over which the plan must be optimal, also known as the horizon, $0 < H \leq \infty$; and s_0 is the starting state of the process. In order to gain insight into the functioning of MDPs, a MDP model of the supply chain scenario, presented in Chapter 1 (see Figure 1.1), will be presented next.

Often, the state of the composition may be factored into a featured set. Each state is then a conjunction of the values of the features. In the supply chain, the state of the composition is captured by the random variables – **Inventory Availability**, **Preferred Supplier Availability**, **New Supplier Availability**, and **Spot Market Availability** A state in the composition is then a conjunction of assignments of either *Yes*, *No*, or *Unknown* to each random variable.

Actions are Web service invocations, A={Check Inventory Status, Check Preferred Supplier Status, Check New Supplier Status, and Check Spot Market Status. Each action typically affects only a subset of the features. When an action is performed at a specific state, the action

[2] I should note that the approaches that will be presented in this thesis may be applicable to any model based service composition technique such as [3] (for example, see [21]).

alters the state, effectively transitioning into a new state. For example, invoking the Web service Check Preferred Supplier Status may cause **Preferred Supplier Availability** to be assigned *Yes*.

The transition function, T, models the non-deterministic effect of each action on some random variable(s). For example, invoking the Web service Check Preferred Supplier Status will cause **Preferred Supplier Availability** to be assigned *Yes* with a probability of T(**Preferred Supplier Availability**=*Yes*|Check Preferred Supplier Status, **Preferred Supplier Availability**=*Unknown*). This is effectively a measure of the rate in which the Preferred Supplier can satisfy order requests. This rate of order satisfaction depends on two probabilities: (1) the probability that the Preferred Supplier has sufficient parts for the order, and (2) the availability of the Preferred Supplier's Web service interface. If the two availabilities are independent of one another[3], we may view T as a product of these two probabilities: T(**Preferred Supplier Availability**=*Yes*|Check Preferred Supplier Status, **Preferred Supplier Availability**=*Unknown*) = Pr(**Preferred Supplier Product Availability**=Yes)× **Web service Availability**. Similarly, the **Preferred Supplier Availability** will be assigned *No* with a probability of T(**Preferred Supplier Availability**=*No*|Check Preferred Supplier Status, **Preferred Supplier Availability**=*Unknown*) = Pr(**Preferred Supplier Product Availability**=No)× **WS Availability**, and *Unknown* with a probability of T(**Preferred Supplier Availability**=*Unknown*|Check Preferred Supplier Status, **Preferred Supplier Availability**=*Unknown*) = 1-**WS Availability**. Note that the latter occurs when the WS fails or is not available.

The cost function, C, prescribes the cost of performing each action. Analogous to the calculation of the transition function T, C is some combination (e.g. a sum) of the cost of invoking the Preferred Supplier Web service and the cost of the part itself. We let H be some finite value which implies that the manufacturer is concerned with getting the most optimal

[3]The two probabilities could be dependent on each other depending on the underlying business logic of the WS. For example, the availability of the part may influence the Preferred Supplier's decision to keep the Web service active. For simplicity in the scenario, however, I assume that they are independent.

WSC possible within a fixed number of steps. Since no information is available at the start state, all random variables will be assigned the value *Unknown*.

We let H be some finite value which implies that the manufacturer is concerned with getting the most optimal Web process possible within a fixed number of steps. Since no information is available at the start state, all random variables will be assigned the value *Unknown*.

Once the manufacturer has modelled its Web service composition problem as a MDP, he may apply standard MDP solution techniques to arrive at an optimal process. These solution techniques revolve around the use of stochastic dynamic programming [56] for calculation of the optimal policy using the Bellman equation (also called *value iteration*):

$$V^n(s) = \min_{a \in A} \; Q^n(s,a) \tag{2.1}$$

where:

$$Q^n(s,a) = \begin{cases} C(s,a) + \sum_{s' \in S} T(s'|a,s) V^{n-1}(s) & \text{n>0} \\ 0 & \text{n=0} \end{cases} \tag{2.2}$$

where the function, $V^n : S \rightarrow \mathbb{R}$, quantifies the minimum long-term expected cost of reaching each state with n actions remaining to be performed, and $Q^n(s,a)$ is the action-value function, which represents the minimum long-term expected cost from s on performing action a.

Once we know the expected cost associated with each state of the process, the optimal action for each state is the one which results in the minimum expected cost.

$$\pi^*(s) = \underset{a \in A}{argmin} \; Q^n(s,a) \tag{2.3}$$

In Eq. 2.3, π^* is the optimal policy which is simply a mapping from states to actions, $\pi^* : S \rightarrow A$. The Web process is composed by performing the WS invocation prescribed by the policy given the state of the process and observing the results of the actions.

Algorithm

Doshi et al. [27] provide an algorithm for the composition, shown in Figure 2.1. It takes the optimal policy, and the starting state of the WSC as input, and interleaves composition and execution. For each state encountered during the execution of the WSC, we refer to the policy of the MDP to recommend the current WS to invoke. The response of the service provides values for the random variables, effectively transitioning into a new state. This process is repeated until H steps are exhausted.

Algorithm for generating WSC
Input: π_n^*, s_0, H
$s \leftarrow s_0$, $n \leftarrow H$
while n > 0
$a \leftarrow \pi_n^*(s)$
Execute Web service a
Get response of a and construct next state, s'
$s \leftarrow s'$, $n \leftarrow n - 1$
end while
end algorithm

Figure 2.1: Algorithm for translating a policy into a WSC.

2.2 Adaptation of Web Service Compositions

The composition approaches described previously are ill-equipped to function properly in volatile environments. As a result, many adaptation approaches have been proposed.

Much of the earlier research in adaptation was inspired by the limitations of early workflow technology. The first workflows were assumed to be more or less an idealized version of the preferred process from which they were modelled. As a result, they were rigidly designed – all components of the workflow (i.e. tasks, roles, data, etc) were static and permanent. If a change in workflow functionality was required, such as a deletion of a task or a change in workflow routing, tedious ad-hoc workflows would have to be created manually and on-the-fly. As a result, process designers and early workflow management systems were ill-equiped

to suitably adapt to sudden changes that may be introduced. Consequently, various research efforts in automatically adjusting workflows to change were spawned to find solutions.

Dynamism has been found to manifest in workflow environments in a variety of ways [61]. Survey works, such as van der Aalst and Jablonski [68], have outlined several types of volatility that may be introduced to these environments. Indeed, the research literature on adaptation is broad.

2.2.1 Traditional Approaches for Adaptation

Most of the earlier work in adaptation focused on handling exceptions that occur in workflows [66, 16, 43]. Exceptions are situations that are not initially modelled by the process and are raised to indicate errors or failures in the workflow during runtime. Such cases include requests to deviate from standard processes due to some external event causing a required task (or service for WSCs) to fail within the workflow. Traditional handlers resolved these failures using manual correction techniques. More recent works, however, have focused on using the event-condition-action (ECA) paradigm, where pre-constructed rules trigger a change in the workflow when exceptions takes place [47]. Typically, transaction constructs are employed, such as task rollbacks and compensations [48, 44]. While exception handlers are crucial components for maintaining properly functioning workflows (and WSCs) in the presence of dynamic environments, they are limited in maintaining long-term, optimal QoS. QoS parameters such as time limits, cost reductions, and availability guarantees were often neglected or overlooked by previous workflow and Web service composition technology in favor of exception-driven architectures, mostly to ensure proper functionality without adding much computational overhead. Additonally, exceptions are commonly viewed as temporary solutions to a fault in the workflow – errors are corrected for individual instances in a case-by-case, ad-hoc manner. Thus, although it remains an important subject of ongoing research, a deeper understanding of process adapatation is required.

Traditional exception handling techniques tended to offer temporary, inflexible solutions to uncommon conditions that occur in workflows and compositions. Newer methods try to construct more permanent solutions, so as to have workflows and compositions evolve in response to changes and make them more self-healing. Researchers began to shift focus to ensuring correctness and consistency in workflows when reacting to a change, with the intuition that some changes in workflows may lead to other unforeseen errors.

The earliest works utilized mathematical models of the entire workflow to ensure consistency. Reichert and Dadam [57] addressed the problem by presenting a formal model of a general workflow in graphical form. From this model, they defined a set of allowable operations that would handle changes to the workflow. These change operations were guaranteed to maintain process correctness and consistency when enacted. Ellis et al. [28] enhances the traditional Information Control Net (ICN) model, a mathematical model designed in the 1970's to model office procedures. Analogous to [57], they model changes mathematically to determine the correctness of a resulting revised workflow. Similarly, Stohr and Zhao [65] devised a Business Process Adaptation Model, which decides how changes in business technologies may affect the needs of an automated workflow.

Other research efforts have tried to identify only the sources of change in a small subset of components in the entire workflow. Van der Aalst et al. [67] addressed the problem of "dynamic change", where systems try to handle old workflow instances in a newly created process following a different business logic, by calculating the "changing region". This region identifies those components of the workflow that are affected by the change in the business process logic, and halts migration to new processes until these components have been enacted. Nanjangud et al. [49] and Charfi et al. [22] apply aspect-oriented programming (AOP) concepts to WSCs to ensure consistency. They extend the BPEL specification by adding support for cross-cutting concerns, which are basically aspects of one part of the process that affects other parts of the process. Desai et al. [24, 25] focus on adapting changing processes using handcrafted protocols (i.e. a set of rules that govern business interaction

interactions). The emphasis of this work is alleviating problems of adapting to changes in business models and policies.

While the previous works make certain guarantees that the workflow will function correctly, little attention was paid to the performance as measured by quality of service (QoS). Only recently have researchers turned their efforts toward identification of change in the individual services' QoS properties [18]. Zeng et al. [81] and Aggarwal et al. [2] were among the first to address process optimality by considering the QoS parameters of services in the selection of optimal workflows, introducing process re-engineering and adaptation at the non-functional level. Paques et al. [52] address changes by creating a WS "adaptation space". The adaptation space represents alternative logical WS compositions that may be used if a previous composition instance fails or is found to be suboptimal. In a similar vein, Chafle et al. [20] present several alternate plans that are pre-specified at the logical level, physical level, and the runtime level of a WSC. Depending on the type of changes in the environment, alternative plans from these three stages are selected. In both cases, adaptation occurs by replacing one serviceable process with another process with similar functionality based on how that process would perform with their current QoS. While capable of adapting to several different events, many of the alternative pre-specified plans are not used making the approaches inefficient. Additionally, neither considers the costs for "switching" from one process to the next. Doshi et al. [27] adapt compositions using a technique that manages the dynamism of WSC environments through Bayesian learning. The process model parameters are updated based on previous interactions with the individual WSs and the composition is regenerated using these updates. This method is slow in updating the parameters, and the approach may result in plan re-computations that do not bring about any change in the WSC.

In a somewhat different vein, Verma et al. [69] and Wu and Doshi [79] explore adaptation in WSCs in the presence of coordination constraints between different WSs. These require that services being invoked concurrently coordinate their behaviors in response to exoge-

nous events that induce volatility in their parameters. Gomadam et al. [31] utilize semantic associations to identify events that may cause changes in a WSC. All of these works are complementary as we do not consider such problems here.

2.2.2 Querying Approaches for Adaptation

While the works above address the adaptation problem in one form or another, many overlook the challenges of querying for revised information as outlined in Section 1.1. Au et al. [8, 9] were among the first to propose a framework that explicitly queries individual component WSs in the presence of data volatility. Their approach was mainly reactionary – if a particular parameter was found to be expired, a query was issued for that parameter and integrated into the WSC model. The model would then recompute a plan for the WSC with the new information. The main drawback of this technique was that plan re-computation is assumed to take place irrespective of whether the revised parameter values are expected to bring about a change in the composition. This may lead to frequent and unnecessary computations. They improved upon this framework by introducing a more intelligent approach called the reactive querying policy [10]. In this framework, WSs parameters were queried and the plan was re-computed if the new parameter information differed from the old parameter information. Harney and Doshi [35] improved upon this approach by introducing the informed-presumptive method, which re-computed the plan only if the new parameter information was expected to change the current plan. While these techniques constitute an important step in understanding the significance of querying individual services for their revised parameters, they still do not address the specific challenges outlined in Section 1.1.

Chapter 3

Motivating Scenarios

In Chapter 1, I introduced a simple supply chain scenario to illustrate the need for adapting WSCs to volatile environments. In this Chapter, I introduce three more scenarios that I will utilize throughout this thesis.

3.1 Microsoft XBOX Supply Chain

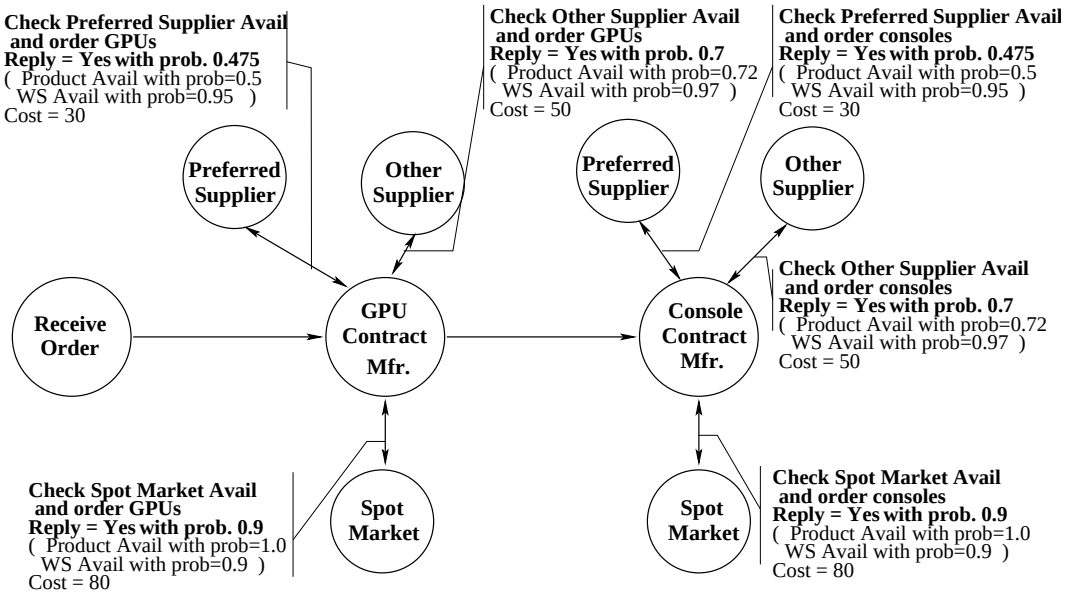


Figure 3.1: Interactions between the business partners in the Microsoft XBox 360 supply chain. Both the contracted manufacturers may select from a choice of suppliers. Example probability and cost values have been included for the purpose of illustration.

Our first scenario is a supply chain employed by Microsoft (MS) for the production of its XBox 360 gaming console [80] designed according to the supply-chain operations reference (SCOR) model [60]. MS engages a variety of suppliers and contract manufacturers to deliver

the components that are crucial to the production of the gaming console. Because MS outsources key manufacturing operations, it needs to retain tight control over those external processes to ensure that the suppliers and contract manufacturers meet service level agreements.

In Figure 3.1, we focus on a simplified supply chain in which MS chooses a contracted console manufacturer, who is responsible for assembling the console, and a contracted graphics processing unit (GPU) manufacturer who is responsible for building the advanced GPU chips. We assume that the invocations will be carried out in a sequential manner, beginning with the GPU followed by the console. Additionally, each of the manufacturers have the option to order their components from three different suppliers. They may order from a preferred supplier with which they usually interact. The manufacturers may also order the parts from other suppliers or resort to the spot market. A *costing analysis* reveals that the least cost will be incurred if the order is satisfied by the preferred supplier. The manufacturers will incur increasing costs as they try to fulfill the orders by procuring the console and GPU chips from another supplier and the spot market.

Clearly, MS and its manufacturers must choose from several candidate compositions. For example, they may initially attempt to satisfy the order of GPU chips from the preferred supplier. If the preferred supplier is unable to satisfy the order, the manufacturers may resort to ordering parts from another supplier. Another composition may involve bypassing the preferred supplier, since MS strongly believes that the preferred supplier will not satisfy the order. It may then initiate a status check on some other supplier. These example compositions reveal two factors for selecting the optimal one. First, MS must accurately know the certainty with which the console and GPU chip orders will be satisfied by each of its supplier choices. Second, at each stage, rather than greedily selecting an action with the least cost, MS must select the action which is expected to be optimal over the long term.

The MSXBOX supply chain is utilized in Chapters 4 and 5.

3.2 Patient Transfer Process

A hospital receives a patient who has complained of a persistent fever. The patient is first checked into the hospital and then seen by one of the hospital's physicians. He may, upon examination, decide to transfer the patient to a secondary care provider for specialist treatment. For this example, we assume that the hospital has a choice of four secondary care givers to select from with differing vacancy rates and costs of treatment, with the preferred one having the best combination of high vacancy rate and least cost (see Figure 3.2).

Similar to our previous example, several candidate processes present themselves. For example, the physician may decide not to transfer the patient, instead opting for in-house treatment. However, if the physician concludes that specialist treatment is required, several factors weigh in toward selecting the secondary care giver. These include the typical vacancy rates of the care givers, costs of treatment, and geographic proximity.

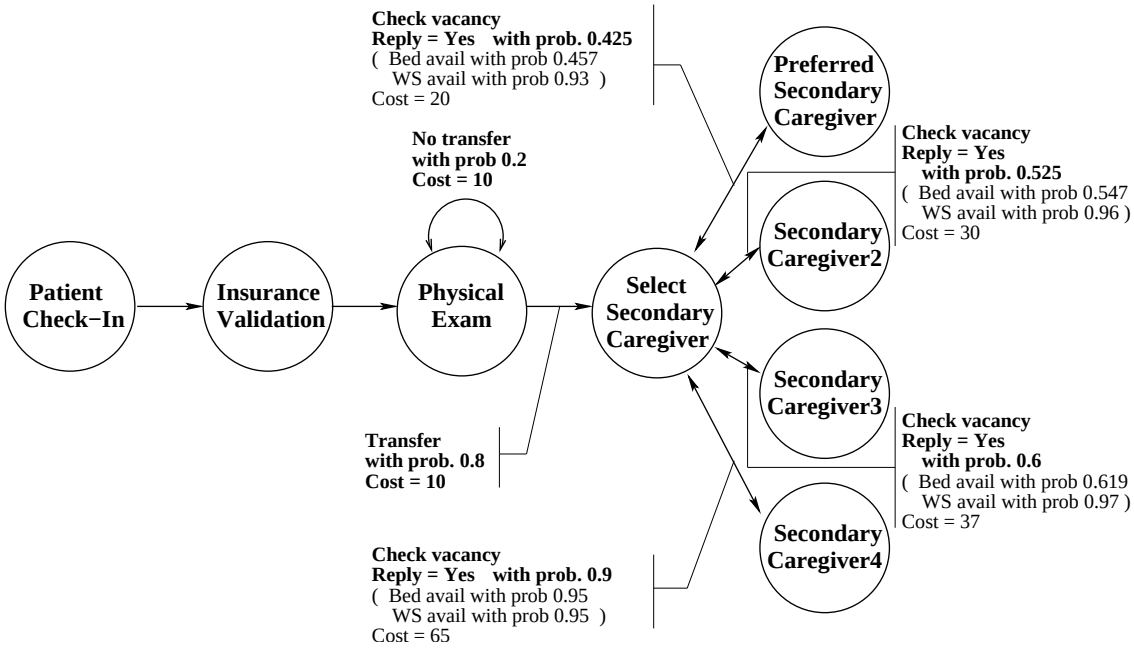


Figure 3.2: The patient transfer clinical pathway for a primary caregiver. If a transfer is recommended, the WSC has a choice of selecting a secondary caregiver WS among many. As before, QoS parameters of the partner services (i.e. probability and cost values) are included for illustration.

The patient transfer process is utilized in Chapters 4 and 5.

3.3 Mortgage Loan Processing

The final scenario is a simplified version of a mortgage loan acquisition process, typically used by brokers that service mortgage loans to individual clients. The broker uses a WSC to automatically process a mortgage loan request. The composition engages a variety of external WSs and vendors to obtain information crucial to securing a suitable home mortgage loan for a client.

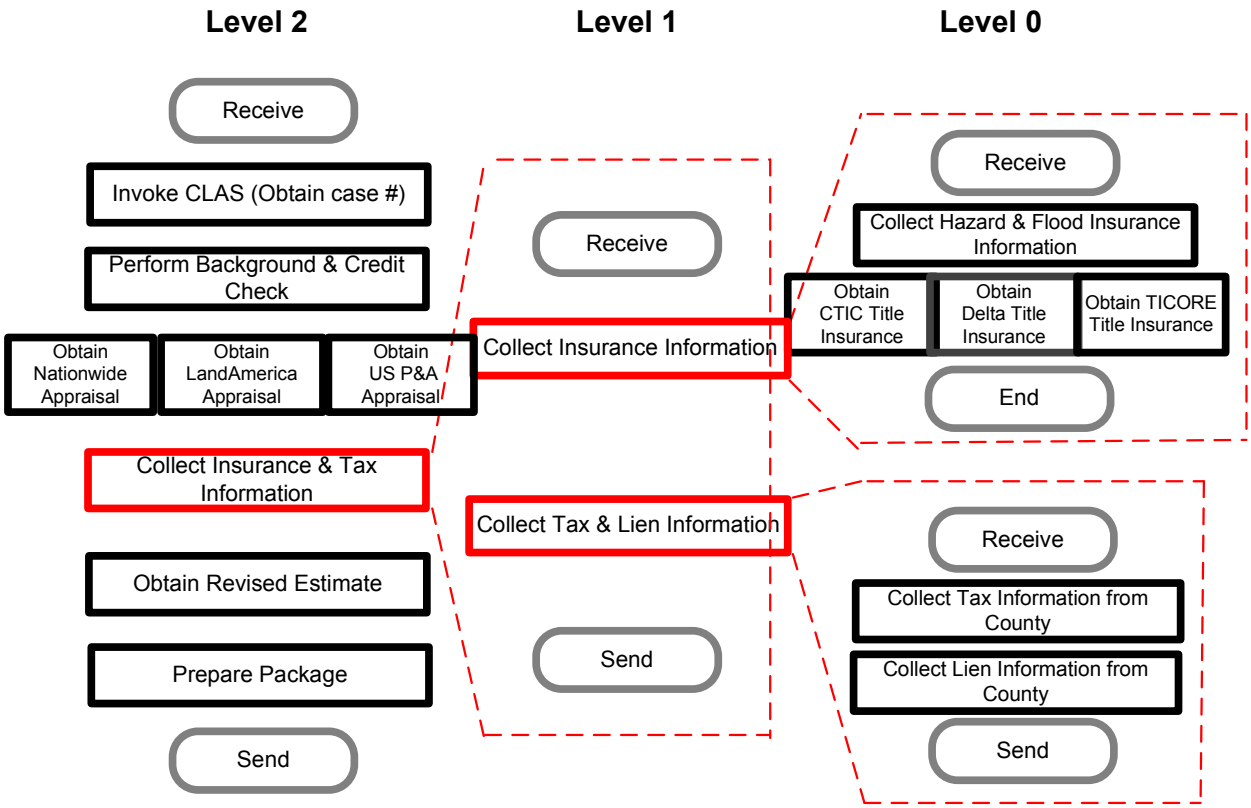


Figure 3.3: A hierarchical mortgage loan process used by a mortgage broker. The broker may choose among multiple appraisal companies and title insurance companies at different levels in the composition.

Figure 3.3 illustrates an example process utilized by the broker. We begin with a description of the upper level of the hierarchy (labeled "Level 2"). The broker receives a request from a client interested in securing a mortgage loan. Some of the activities that the broker performs are to issue a request to the CLAS (CHUMS Lender Access System) WS to obtain

a case number for this particular client, utilize the WS of the credit rating agencies to perform financial background and credit checks of the client, and hire an appraisal agency to appraise the value of the home. In this example, the broker may choose between three different appraisal services: Nationwide, LandAmerica, and US P and A. A choice among these is made depending on the cost and the availability of using the service at a given time. If a chosen service is unable to perform the appraisal, then one of the other appraisal services may be used instead. Next, the broker invokes the insurance and tax information collection service to obtain vital home insurance, tax and lien information. With all of this information on hand, the broker will complete an estimate of the loan and prepare the complete package for the client to review.

We formulate the Insurance and Tax Information service (shown in red) as a composite service. The corresponding lower level (level 1) composition consists of two services for collecting insurance information and collecting tax and lien information. These services are themselves composite. The Insurance Information service involves gathering information on both hazard and title insurances. Note that at this stage, the broker must decide on a vendor (CTIC, Delta, or TICORE) to provide the title insurance. Finally, the Tax and Lien Information service invokes home county services to extract the tax and lien information on the home. The costs incurred to the broker hinge, in part, on the probability with which the appraisal services and the title insurance services may process a request. For example, if the request completion rate of a previously chosen appraisal service (such as Nationwide) suddenly drops, the WSC must adapt (ie. utilize a different service) to remain cost-effective.

The mortgage broker process is utilized in Chapter 6.

3.4 Summary

The scenarios presented represent a wide range of application domains. All of the scenarios, however, share a motivation to remain optimal in the presence of a volatile environment. Each of the scenarios require up-to-date knowledge of the revised parameters to remain cost

effective. For example, the effectiveness of the MSXBOX supply chain hinges, in part, on knowning the up-to-date rates of order satisfaction. If the order satisfaction rate of a supplier suddenly drops, the WSC must become aware of this change so that it may adjust its model (perhaps by choosing a different supplier) to remain cost-effective. The effectiveness of the other scenarios are determined analogously. The approaches presented in this thesis will address these concerns.

Chapter 4

Selective Querying Using the Value of Changed Information

In order to adapt optimally to volatile environments, compositions must possess up-to-date knowledge of the revised parameters of the services that participate in the composition. We have seen in Section 2.2.2 that these changes may be monitored by *querying* component services for their revised parameter values. The changed values can then be integrated into the composition model so that it may make more informed, and consequently more optimal, selections of candidate Web services for participation in the composition. Not all updates to the model parameters, however, cause changes in compositions. Furthermore, the change effected by the revised information may not be worth the cost of obtaining it. In light of these arguments, a method is needed that will suggest a query, only when the queried information is *expected* to be sufficiently valuable. In this chapter, I introduce a novel approach that addresses these concerns by intelligently querying component WSs. I call this approach, selective querying using the *Value of Changed Information (VOC)*. Section 4.1 formally defines the VOC methodology and the concepts from which it originates. Section 4.2 demonstrates the algorithm utilized for adapting Web service compositions using the VOC. Section 4.3 outlines the architecture required to implement a VOC-based methodology. Section 4.4 provides empirical evidence of the improved performance of WSCs that utilize the VOC-based querying strategy over other WSCs that employ other state-of-the-art querying approaches.

4.1 VOC Definition

QoS parameters of participating services may be subject to change during the life-cycle of a WSC. For example, the cost of using the GPU Preferred Supplier's service in the MSXBOX

supply chain scenario (Section 3.1) may increase or the probability with which the GPU Preferred Supplier meets the orders may reduce. The former requires an update to the cost function component, C, while the latter requires an update of the transition function component, T [1].

In lieu of querying all services at any given time step and updating a composition model with all revised information at once, I adopt a *myopic* approach to information revision, in which a single provider is queried at a time for revised information. In the MSBOX supply chain, this would translate to asking, say, only the GPU Preferred Supplier for its current rates of order satisfaction, as opposed to both the GPU Preferred Supplier and the Inventory WSs, simultaneously. For example, the revised information may change the following transition probabilities, T(**GPU Preferred Supplier Availability** = Yes | Check GPU Preferred Supplier Status, **GPU Preferred Supplier Availability** = Unknown), T(**GPU Preferred Supplier Availability** = No | Check GPU Preferred Supplier Status, **GPU Preferred Supplier Availability** = Unknown), and T(**GPU Preferred Supplier Availability** = Unknown | Check GPU Preferred Supplier Status, **GPU Preferred Supplier Availability** = Unknown). Myopic approaches are based on the same heuristic idea as greedy search found in many classical algorithms. These approaches to information revision have been found to be very powerful and work well in practice [59]. They are also more practical in a SOA setting because they are computationally efficient. While non-myopic approaches have been proposed for information revision, they are not yet well-developed and are a topic of future study.

Since the actual revised transition probability is not known unless we query the service provider, we must average over all possible values of the revised transition probability, using our current belief distributions over the values. These distributions may be provided by the service providers through pre-defined service-level agreements (SLAs) or they could be

[1] In this thesis, I only address volatility in the availability and cost parameters of WSs. The approach, however, may be generalized to accomodate fluctuations in other important quantitative parameters such as WS response time and reliability. Changes in qualitative properties such as WS security and trust, however, is out of scope.

learned from previous interactions with the service providers. These beliefs may depend on any number of economic factors. For example, Microsoft's beliefs of the GPU Preferred Supplier's rate of order satisfaction may depend on the quantity of the order or the time or season of the order (e.g. holiday seasons leading to a higher demand of its consoles).

We may model the beliefs using *probability density functions* (pdfs). For example, the manufacturer for the supply chain in Figure 1.1 models its beliefs using beta density functions. Figure 4.1 shows the beta densities that represent the manufacturer's distribution over the rate of order satisfaction by, say, the Preferred Supplier, i.e. Pr(Preferred Supplier Availability = Yes — Check Preferred Supplier, Preferred Supplier Availability = Unknown), and analogously for the other suppliers.

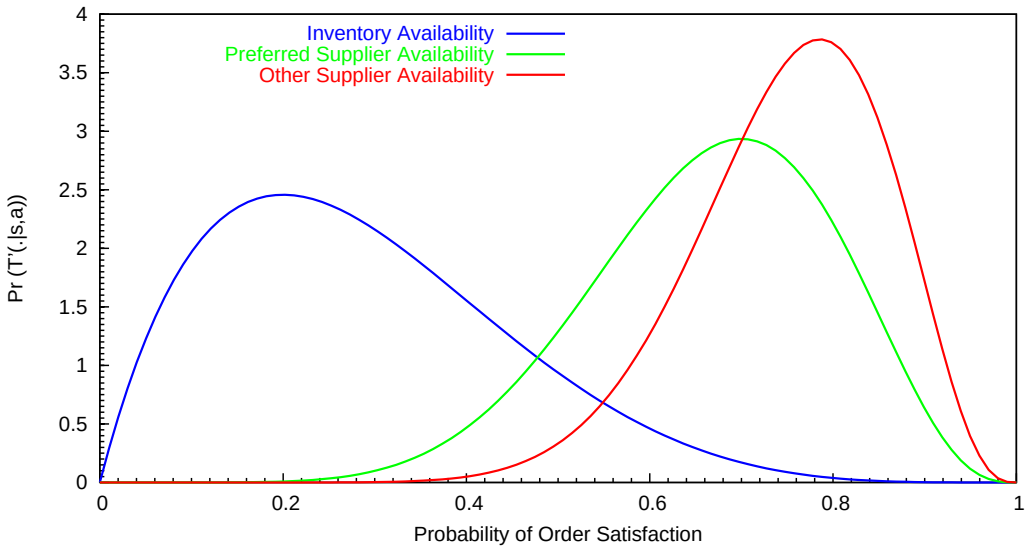


Figure 4.1: The probability density functions representing the manufacturer's belief over the suppliers' rates of satisfaction in the supply chain scenario in Chapter 1.

These densities are projections of the more complex ones that would account for all the factors that may influence a supplier's ability to satisfy an order. Furthermore, they reveal important information about the service providers and their ability to satisfy orders. For example, Inventory tends to be less reliable in satisfying orders than the other suppliers.

Following the MDP model for composition in Section 2.1.2 and the Bellman equation in Eq. 2.1, let $V_{\pi^*}(s|T')$ denote the expected cost of following the optimal policy, π^*, from the state s when the revised transition function, T' is used.

Formally,

$$EV(s) = \int_{\mathbf{p}} Pr(T'(\cdot|a, s') = \mathbf{p}) V_{\pi^*}(s|T') d\mathbf{p} \tag{4.1}$$

where $T'(\cdot|a, s')$ represents the distribution that may be queried and subsequently may get revised, $\mathbf{p} = \langle p_1, p_2, \ldots, p_m \rangle$ represents a possible response to the query (revised distribution), m is the number of values that the variable under question may assume, and $Pr(\cdot)$ is our current belief over the possible distributions.

As a simple illustration, suppose that the Preferred Supplier is queried for its current rate of order satisfaction. Eq. 4.1 becomes,

$EV(s) = \int_{\langle p_1, p_2, 1-p \rangle} Pr(T'$ (**Preferred Supplier Availability**=Yes/No/Unknown |Preferred Supplier Status, **Preferred Supplier Availability** = Unknown)= $\langle p_1, p_2, 1 - (p_1 + p_2) \rangle)$ $V_{\pi^*}(s|T')d\mathbf{p}$

assuming that the random variable **Preferred Supplier Availability** assumes either *Yes*, *No*, or *Unknown* on checking the status of the Preferred Supplier.

Let $V_{\pi}(s|T')$ be the expected cost of following the original policy, π, from the state s in the context of the revised model parameter, T'. Note that the policy, π, is optimal in the absence of any revised information. The *value of change* (VOC) due to the revised transition probabilities is formulated as:

$$VOC_{T'(\cdot|a,s')}(s) = \int_{\mathbf{p}} Pr(T'(\cdot|a, s') = \mathbf{p}) [V_{\pi}(s|T') - V_{\pi^*}(s|T')] d\mathbf{p} \tag{4.2}$$

where the subscript to VOC, $T'(\cdot|a, s')$, denotes the revised information inducing the change. Intuitively, Eq. 4.2 represents how *badly, on average, the original policy, π, performs in the changed environment* as formalized by the MDP model with the revised T'.

VOC shares its conceptual underpinnings with the value of perfect information (VPI) [59]. Indeed, both of them may be seen as special cases of the value of information idea, which determines whether new information is useful to a particular WSC. However, there is an important difference between the two concepts. VPI computes the value of *additional* information, while the VOC provides the value of *revised* information. This distinction is illustrated using the following example:

Example In the supply chain example, the VPI provides a way to gauge the expected impact of knowing additional (previously unknown) parameters of Web services such as say, time to service failure and time to service repair, on the composition. In comparison, the VOC measures the expected impact of revised values of parameters that were previously considered while forming the initial composition, such as order satisfaction rate and service cost.

Analogous to VPI, the following proposition holds for VOC.

Proposition 4.1.1 $\forall s \in S, \quad VOC(s) \geq 0$ *where VOC(·) is as defined in Eq. 4.2.*

Proof The proposition follows trivially if we find that $\forall s, \mathbf{p} \quad V_{\pi}(s|T') - V_{\pi^*}(s|T') \geq 0$, where $T'(\cdot|a, s') = \mathbf{p}$. By definition (Eq. 2.3), π^* is an optimal policy for the revised model. This implies that for any other policy, $\pi' \in \Pi \backslash \pi^*$, where Π is the space of all policies, $\forall \mathbf{p} \quad V_{\pi'}(s|T') \geq V_{\pi^*}(s|T')$. This holds true over all the states. The required inequality obtains since π must either be in $\Pi \backslash \pi^*$, or be equal to π^*.

It is important to note that the proof of Theorem 4.1.1 assumes that the revised information is received *without* noise. Integrating noisy, or inaccurate, information into a composition model may lead to compositions that do not act optimally in volatile environments. It is therefore critical to use the VOC methodology with trusted service partners who provide accurate measurements of their QoS properties. While measuring trust is beyond the scope of this thesis, it is an important complement that should be considered in the future.

As we are using a myopic approach for information revision, we select the service provider associated with the Web service invocation, a, to possibly query for whom the VOC is maximum:

$$a^* = \underset{a \in A}{argmax} \;\; VOC_{T'(\cdot|a,s')}(s) \tag{4.3}$$

Let VOC*(s) represent the corresponding maximum VOC. We may then obtain VOC*(s) as follows:

$$VOC^*(s) = \underset{a \in A}{max} \;\; VOC_{T'(\cdot|a,s')}(s) \tag{4.4}$$

Querying for information from service providers may often tedious, time consuming, and subsequently, expensive. The expenses could include, for example, contractual costs and intangible costs such as the delay incurred while awaiting the revised information. Thus, querying should only be undertaken if it is expected to pay off. In other words, the revised information is queried in that state of the composition only if the VOC due to the revised information in that state is greater than the query cost. More formally, if,

$$VOC^*_{T'(\cdot|a,s')}(s) > QueryCost(T'(\cdot|a, s'))$$

where $T'(\cdot|a, s')$ represents the distribution we want to query, a query is issued to WS a.

The VOC methodology may analogously applied when the cost parameters of the services fluctuate. Instead of updating the transition function, T, we update the cost function, C. We obtain the cost distribution that may be queried, $C'(a, s)$, from the provider associated with WS a and use this distribution to find both $V_{\pi^*}(s|C')$ and $V_{\pi}(s|C')$. We may then use Eq. 4.2 analogously to find $VOC_{C'(a,s)}(s)$ and query if $VOC_{C'(a,s)}(s) > QueryCost(C'(a, s))$.

4.2 VOC Algorithm

Only a small revision to the composition algorithm in Figure 2.1 is necessary in order to utilize the VOC-based querying methodology for adapting WSCs. The revised algorithm for an adaptive composition is given in Fig. 4.2.

Algorithm for adaptive WSC Using VOC
Input: π_n^* *//optimal policy*, s_0 *//initial state*, H *// Horizon*
$s \leftarrow s_0$, $n \leftarrow H$
while n > 0
if VOC*(s) $>$ QueryCost$(T'(\cdot|a, s'))$
Query service provider, a^* (Eq. 4.4), for new probabilities
Form the new transition function, T'
Calculate policy π_n^* using the new MDP model with T'
$a \leftarrow \pi_n^*(s)$
Execute the Web service a
Get response of a and construct next state, s'
$s \leftarrow s'$, $n \leftarrow n - 1$
end while
end algorithm

Figure 4.2: Algorithm for adapting a WSC to revised information.

For each state encountered during the execution of the WSC, we query a service provider for new information if the query is expected to bring about a change in the WSC that exceeds the query cost. For example, in the supply chain WSC, we select and query a supplier for its current rate of order satisfaction.

4.3 Service Oriented Architecture with VOC

The algorithm described in Fig. 4.2 may be implemented within an SOA as a WS-BPEL flow. To the WS-BPEL flow, we give the optimal policy, the start state, and horizon as input. An outline of the SOA is shown in Figure 4.3.

Within the SOA, internal WSs are provided for generating the policy from the MDP model and for computing the VOC. If the VOC exceeds the cost of querying a provider (this cost is also provided as an input), the WS-BPEL flow invokes a special WS whose function is to query the provider's information-providing WSs for revised information. This information is used to form and solve a new MDP. The new policy is fed back to the WS-BPEL flow. The policy is used by the composition to invoke the prescribed external provider WS. The WS

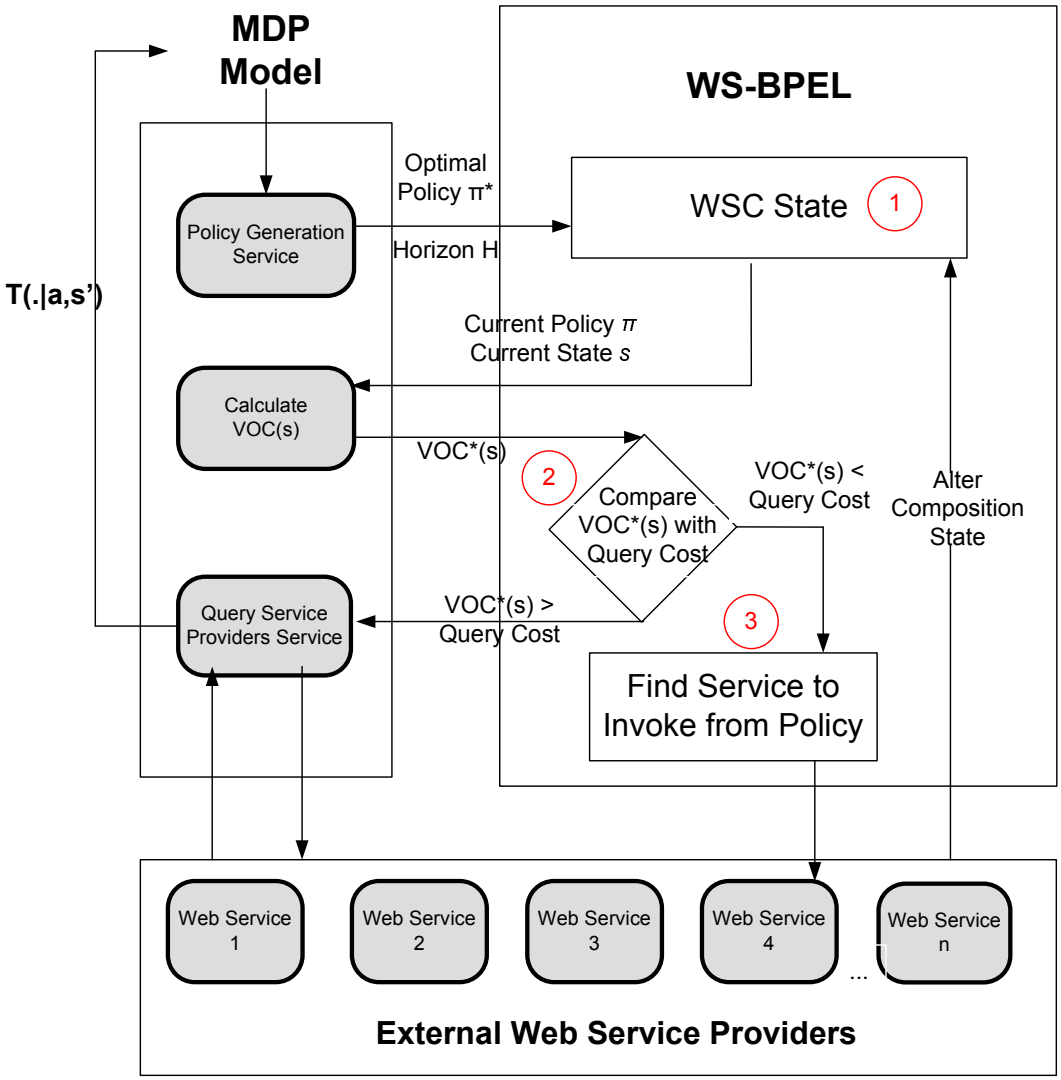


Figure 4.3: SOA for implementing our adaptive WSC demonstrating the interaction of the composition with our pre-constructed internal services. Labels (1) (2) and (3) correspond to its associated markup in Figure 4.4.

response is used to formulate the next state of the composition. This procedure is repeated until H steps have been exhausted.

As WS-BPEL is utilized in a somewhat non-standard way, details of the WS-BPEL markup is given in Fig. 4.4. First, note that the following constructs must be added to implement the VOC algorithm:

BPEL Markup for MS XBox Supply Chain

```
<!-- Place namespaces here -- >
<Variables>
<!-- state and policy information -- >
  <variable name="stateData" messageType="stateMessage" />
  <variable name="policy" messageType="policyMessage" />
  <variable name="horizon" messageType="horizonMessage" />
<!-- Suppliers' variables -- >
  <variable name="CGPUPSreq" messageType="GPUPreferredSupplierRequestMessage"
  <variable name="CGPUPSres" messageType="GPUPreferredSupplierResponseMessage"/>
<!-- Place remaining suppliers here -- >
<!-- Place internal service (VOC and Policy Generator variables here -- >

   </Variables>
<!-- Place partnerlinks here-- >
<sequence name="MSXBOX Supply Chain Scenario">
  <receive partnerLink="…" portType="…" operation="start WSC" variable="state" />
<!-- Loop until the goal state is reached -- >
  <while condition="bpws:getVariableData('horizon', 'steps, '//numSteps') &gt; 0">

<!-- Find the state by using the instance variables -- >
   <switch>
     <case condition="bpws:getVariableData('stateData', 'stateData', '//CGPUPS') = 'unknown'  and
                      bpws:getVariableData('stateData', 'stateData', '//CGPUOS') = 'unknown'  and
                      bpws:getVariableData('stateData', 'stateData', '//CGPUSM') = 'unknown'  and
                      bpws:getVariableData('stateData', 'stateData', '//CConsolePS') = 'unknown' and
                      bpws:getVariableData('stateData', 'stateData', '//CConsoleOS') = 'unknown' and
                      bpws:getVariableData('stateData', 'stateData', '//CConsoleSM') = 'unknown'>

<!-- Invoke the VOC Calculator -- >
      <invoke name="invoke" partnerLink="..." portType="..."  operation="..."
              inputVariable="VOCreq"  outputVariable="VOCres">  </invoke>
<!-- get response  and compare with Query Cost -- >
     <assign>  <copy>  <from expression="'VOCresponse'"/>  <to variable="VOCres"/>  </copy>  </assign>

     <switch>
       <case condition="bpws:getVariableData('VOCres', 'VOCCalculatorReturn' ) &gt;
                        bpws:getVariableData('stateData','stateData','queryCost')">
<!-- if greater than query cost, query the service and invoke policy regeneration service for new policy -- >

<!-- get the service to invoke from the policy
     and invoke that service -- >
      <switch>
        <case condition="bpws:getVariableData('policy','policy','//initialState') = 'invoke GPUPS">
          <invoke name="invokeCheckGPUPSStatus" partnerLink="..." portType="..." operation="CheckStatus"
                  inputVariable="GPUinput" outputVariable="GPUoutput">
          </invoke>
          <!-- Note: it could be any of the 6 services -- >

<!-- get response and reassign the state -- >
      <assign>  <copy>  <from expression="'GPUoutput'"/>  <to variable="stateData" part="GPUPS"/>
                </copy>  </assign>
    </case>
<!-- decrement the number of steps  -- >
<!-- repeat for all states -- >
  </while>
  <reply partnerLink="…" portType="…" operation="end WSC" variable="stateData" />
</sequence>
```

Figure 4.4: A sample of the BPEL markup for the MS XBox supply chain scenario. Labels (1) (2) and (3) correspond to its component in Figure 4.3.

- data structures for state and policy,
- tasks invoking a VOC computation service, comparing the VOC to the query cost, and regenerating the policy
- a task that will invoke the external services providers as recommended by the policy.

As outlined by section (1) in Fig. 4.4, state is stored in the BPEL document by creating a complex message type, $stateMessage$ and stored in the $stateData$ variable. Similarly, complex message type $policyMessage$, is stored in the $policy$ variable, and used to represent the given policy.

The $< while >$ condition corresponds to the while loop in Fig. 2.1. Each state has an associated $< switch > < case >$ construct [2]. In each $< case >$, the WSC invokes the VOC WS, which upon completion, returns VOC^* (section (2)). The VOC^* value is compared to a $QueryCost$ variable. If the returned VOC^* is greater than the $QueryCost$, then the associated service is queried. The queried parameters are integrated into the MDP, which invokes the policy generator service, returning the new optimal policy thereby replacing the old policy of the WSC. The new policy is then used to recommend the optimal service to invoke in the state (section (3)). This process repeats until the composition has terminated. I have included a more detailed discussion of the BPEL markup in the Appendix [3].

4.4 Performance Evaluation

To demonstrate the improvements of VOC, let us initially compare the performance of the VOC-based querying approach for adaptation with other naive methods of querying for the simple supply chain outlined in Chapter 1. The first method assumes that there is no adaptation to the volatile environment and uses a statically created policy for every execution of

[2]Note that in the most recent BPEL specification (BPEL 2.0 [15]), the <switch> statement has been replaced by <if> to represent conditional statements in compositions.

[3]Note that there may be some inconsistency in some of the constructs in the Appendix and Figure 4.4 due to the migration from BPEL specification 1.1 [14] to BPEL specification 2.0 [15]

the composition. A MDP model is formulated and solved before the first execution instance and the resulting policy is used for every instance then onwards. The second method implements a periodic querying strategy, in which a service provider is selected at random and queried for revised information at each state of the WSC. Using the new information, the policy is re-solved and the composition continues to run using the new policy. Finally, the VOC is employed. If the VOC determines if the WSC should query, it will issue the query, re-solve the policy and continue to run using the new policy.

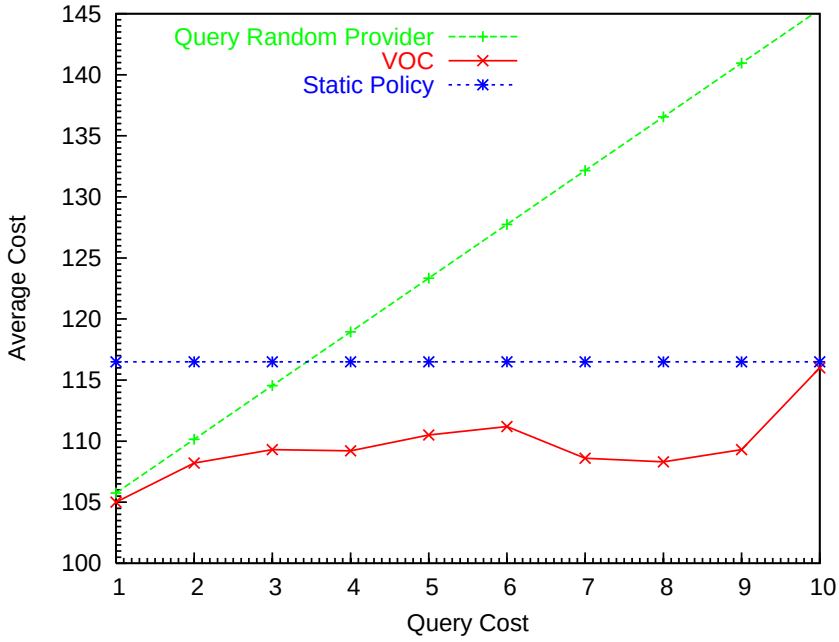


Figure 4.5: Comparisons of the VOC based composition with the static policy and periodic querying approaches for adapting the simple supply chain.

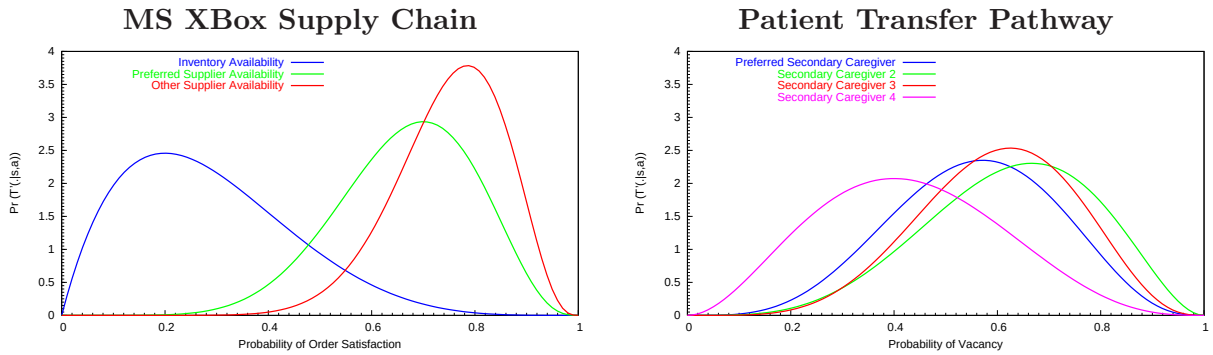
In Fig. 4.5, the three strategies are compared with respect to the average cost incurred from the execution of the composition as the cost of querying the service providers is increased. Using the architecture defined in Figure 4.3, the experiment consisted of running 500 independent instances of each composition within a simulated volatile environment, where the parameters of the service providers were distributed according to the density plots in Fig. 4.1. The compositions using each of the three strategies received similar responses from the service providers, but these responses may change between instances. The experiments utilized ActiveBPEL's BPEL engine [1] for executing the BPEL process and the

WSs were implemented in the AXIS2 [6] container and exposed on the Apache Tomcat [7] web server. Each of the provider services are assumed to have an associated service-level agreement [17] exposing their QoS attributes.

Intuitively, as we increase the cost of querying, our VOC based approach performs less queries and thus adapts the WSC less. For a large query cost, its performance is similar to a WSC that does not change its policy. In Figure 4.5, we show the results for the supply chain scenario. For smaller query costs, a VOC based approach will query frequently, though not as much as a strategy that always queries (query always). Hence, the average cost is close to that incurred for the latter. As we increase the query costs, the VOC based approach will allow a query for revised information only if its value exceeds the cost. Thus, a composition that is adapted using VOC performs better (incurs less average cost) in a volatile environment because only significant changes are carried out while simultaneously avoiding frequent costly queries. A strategy that queries periodically adapts well to a volatile environment only when the cost of querying is small. As the query costs grow larger, the average composition cost will steadily increase.

Let us now utilize the MS XBox supply chain and the clinical patient transfer scenarios (Sections 3.1 and 3.2) for evaluation. Analogous to the simple supply chain, we model the MS-XBOX manufacturer and primary caregiver's beliefs over the possible parameters of the service providers, ($Pr(T'(\cdot|a, s') = \mathbf{p}$) in Eq. 4.2) using beta density functions. Figure 4.6(a) shows the beta densities that represent the MSXBOX manufacturer's distribution over the rate of order satisfaction of the supplier services and Figure 4.6(a) shows the densities for the secondary caregivers in the patient transfer scenario.

We now compare the VOC-driven selective querying based adaptation with four other strategies with respect to the average cost incurred from executing the adapted WSCs, as the cost of querying the service providers is increased. The four other approaches utilized for adaptation are:



(a) (b)

Figure 4.6: The probability density functions representing (a) MS's belief over the GPU and console suppliers' rates of order satisfaction in the XBox supply chain scenario; (note that both the GPU and Console Suppliers have the same curves because their parameters for our experiments were identical) (b) the primary caregiver's beliefs over the secondary caregivers' probabilities of having a vacancy.

1. **Static policy** This is our baseline approach that ignores adaptation and the initial policy is utilized unchanged for executing the composition in each instance.

2. **Random query** In this approach, we randomly select a service each time for querying for revised information.

3. **Intermittent querying** We begin by querying services every alternate instance. As the costs of querying increase, we reduce the frequency with which we query.

4. **Largest difference** This approach utilizes the distributions of the services parameters shown in Fig. 4.6. It selects a service to query whose existing parameter value is most different from the mean as obtained from the corresponding distribution.

Note that each of the approaches mentioned above are naive analogies of certain aspects of the VOC based approach. Thus, the approaches provide an effective testbed with which to compare our VOC based WSC adaptation.

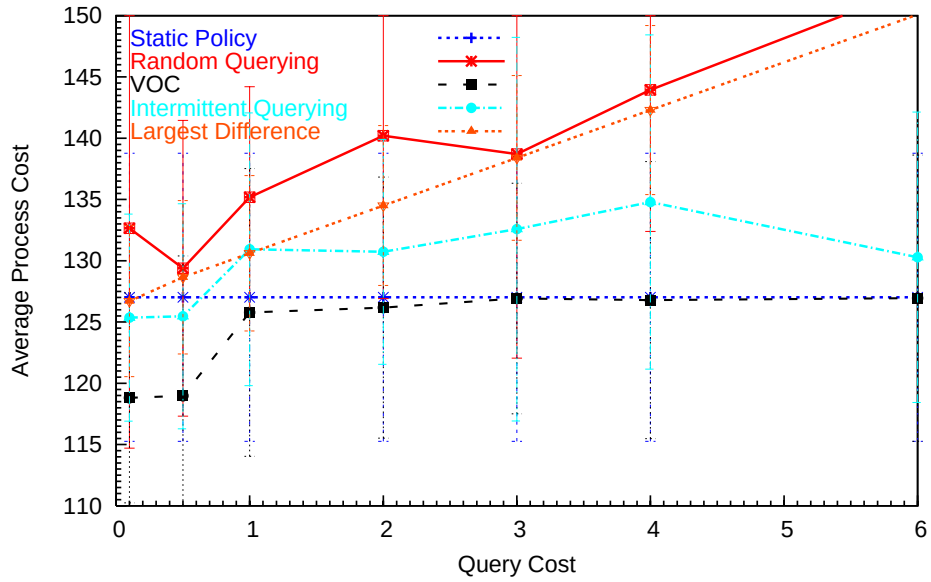
Once again, a trial of 500 independent instances of each composition was executed within a simulated volatile environment, where the queried parameters of the service providers

were distributed according to the density plots in Fig. 4.6. The compositions using each of the five strategies received similar responses from the service providers. For the MSXBOX supply chain example, we simulated querying the suppliers for their current percentage of order satisfaction while in the patient transfer clinical pathway, we queried the secondary caregivers for their current vacancy rates. The costs of each of the provider WSs are given in Figure 4.7.

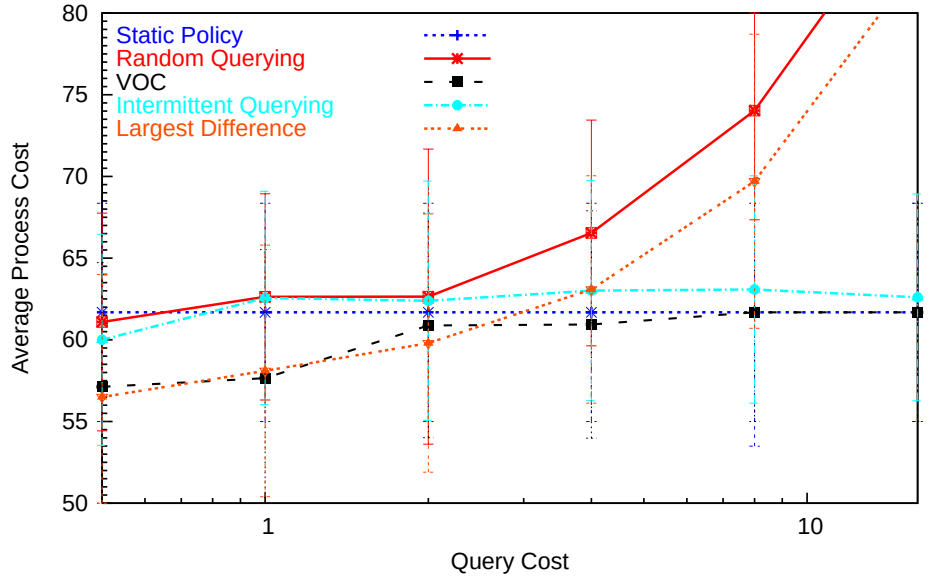
MS XBox Services	**Cost**
GPUPreferredSupplier	30
GPUOtherSupplier	50
GPUSpotMarket	80
ConsolePreferredSupplier	30
ConsoleOtherSupplier	50
ConsoleSpotMarket	80
Patient Transfer Services	**Cost**
InsuranceValidation	10
PhysicalExam	10
PreferredSecondaryCaregiver	20
SecondaryCaregiver2	30
SecondaryCaregiver3	37
SecondaryCaregiver4	65

Figure 4.7: Costs for the WSs in the MS XBox supply chain and Patient Transfer scenarios.

In Figures 4.8(a) and (b), the results for the MSXBOX supply chain and patient transfer scenarios are respectively demonstrated. For smaller query costs, a VOC based approach will query frequently, though not as much as a strategy that always queries a provider, such as *random query*. As we increase the query costs, the VOC based approach will allow a query for revised information only if its value exceeds the cost. Though, *intermittent querying* naively seeks to emulate this behavior, it performs worse because it does not utilize the value of a potential change in the composition in deciding when to query. We note that the *largest difference* approach performs well for lower query costs (in particular, see Fig. 4.8(b)), though worse than the VOC based approach. This is because the service exhibiting the largest difference from the mean in its parameter value is often the one that brings about the largest change in the composition. However, this is not always the case – for example, a large change



(a)



(b)

Figure 4.8: Comparisons of the VOC based adaptive WSC with the static policy and other querying approaches for (*a*) MS XBox supply chain, and (*b*) patient transfer scenarios. Lower average process cost indicates better performance. The deviation bars demonstrate the variance caused by a randomized environment.

in the parameter of a mandatory service, such as the *physical exam* service in the patient transfer composition, does not affect the composition though the approach will query it and incur the query cost. In addition, the difference in parameter values is not comparable to query costs. In summary, a composition that is adapted using VOC performs better (incurs less average cost) because only significant changes to the composition are carried out while simultaneously avoiding frequent costly queries.

4.5 Summary

QoS parameters of the service providers may be subject to change over time. In such environments, WSCs should query for revised information in order to adapt and remain cost effective. The revised information, however, must be carefully obtained, as querying may be expensive. I presented a selective querying strategy for information revision called the *value of changed information* (VOC), which intelligently adapts a WSC to changes in certain parameters of the service providers. Specifically, the approach measured the expected value of change that the revised information may bring to the WSC and compares it with the cost of obtaining the information. If the revised information is worth the cost of obtaining it, we query the service providers for their current parameters and reformulate the composition using the revised information. As a result, when VOC-based querying was applied to adaptive WSCs, it was found to be a more effective approach than other state-of-the-art methods.

Chapter 5

Mitigating Complexity of the Value of Changed Information

Chapter 4 demonstrated that we may improve average costs in executing WSCs in the presence of volatile environment by utilizing the VOC-based querying methodology for adaptation. However, this improvement comes at a computational price. In Figure 5.1, the average runtimes of executing and adapting the MSXBOX supply chain and patient transfer compositions are presented. Here, we see that an adaptive composition that uses VOC takes an order of magnitude more than the adaptive composition that uses a random querying method. In SOA environments where turnaround time is of utmost importance, we must find ways to mitigate the computational complexity of VOC so that it becomes a more feasible approach for adaptation.

Problem	**Query Random**	**VOC**
Supply Chain	17s	285s
Patient Transfer	15s	101s

Figure 5.1: A comparison of the average execution times of the MS XBOX supply chain and patient transfer compositions that utilize a random querying strategy and the VOC strategy for adaptation.

In this chapter, I present techniques that alleviate the complexity of computing VOC. In Section 5.2, I observe that for particular values of the parameters, the composition remains unchanged. Consequently, such revised values may be ignored, as they do not change the composition. I provide a simple and quick way to ascertain these values. In Section 5.3, I utilize the expiration times often associated by service providers to the parameters of their services, using the intuition that I need not consider querying those services (or associated information providers) for revised information whose previously obtained information has

not expired. I incorporate this insight into the VOC formulation, and call the approach, the *value of changed information with expiration times* (VOC$^{\mathcal{E}}$). Because VOC$^{\mathcal{E}}$ focuses the computations on only those services whose parameters could have changed, it is computationally more efficient than the traditional VOC.

5.1 Computational Complexity of VOC

Calculating the VOC as shown in Eq. 4.2 is a computationally intensive procedure. The probability $\mathbf{p}$ represents a revised probability of transition on performing a particular action. To calculate the VOC, I must compute the revised values, $V_{\pi}(s|T')$, and $V_{\pi^*}(s|T')$ for *all possible* $\mathbf{p}$ and average over their difference based on our distribution over $\mathbf{p}$. Computing $V_{\pi}(s|T')$, which represents the expected cost of following the policy π from state s is straightforward since it does not involve the minimization operation. However, the revised value function $V_{\pi^*}(s|T')$ is computed using the Bellman equation (Eq. 2.1), which is expensive because it recomputes policy π^*. Specifically, if N is the number of possible values a random variable X^k can take, then the complexity of the Bellman equation is $\mathcal{O}(N^{2|X|}|A||H|)$. However, recomputing the entire policy is necessary in general, as any state may be subsequently reached in the WSC, including previously visited states.

Furthermore, calculating the VOC* as shown in Eq. 4.4 is computationally intensive. It involves iterating over all the service providers and computing the VOC for each. Because there could be many service providers participating in the composition, a more selective approach is needed to obtain computational efficiency.

5.2 Speeding Up Adaptation Using Pruned Averaging

One way of mitigating the computational cost of calculating VOC is to reduce the range of $\mathbf{p}$ over which the averaging is carried out. In order to reduce the effective range of $\mathbf{p}$, and more generally the effective range of possible values of parameters, we may find out those values of $\mathbf{p}$ for which the expected value of the optimal policy given the revised probability,

$V_{\pi^*}(s|T')$, remains unchanged from that of the existing policy. This typically occurs because the revised parameters did not alter the existing optimal policy. These values of $\mathbf{p}$ do not contribute to the VOC and therefore may be ignored without affecting the VOC. Obviously, there is at least one point in the range of $\mathbf{p}$ which does not contribute to the VOC and may be pruned. Proposition 5.2.1 formalizes this observation:

Proposition 5.2.1 *Given state s of the composition guided by policy π, and WS a, there exists at least one $T'(\cdot|a, s') = \mathbf{p}'$, for which $V_{\pi^*}(s|T') = V_{\pi}(s|T')$.*

Proof Let $\mathbf{p}'$ be the current value of the parameter of the composition under consideration. Given that all other parameters remain unchanged, the optimal policy, π^*, when $T'(\cdot|a, s') = \mathbf{p}'$ is also the current policy. Therefore, $V_{\pi^*}(s|T') = V_{\pi}(s|T')$.

Although Proposition 5.2.1 points to the existence of at least one $\mathbf{p}$ in the integral range of Eq. 4.2 that does not contribute to the VOC, typically this range tends to be larger. For example, in Figure 5.2, I show the plots of $V_{\pi^*}(s|T')$ and $V_{\pi}(s|T')$ as $\mathbf{p}$ is varied, for the supply chain problem where the WS under consideration is GPU Preferred Supplier in its initial state (s_0). Notice a significantly large region of overlap between the two plots – the range of $\mathbf{p}$ spanning the overlap does not contribute to the VOC and may be ignored. As an aside, observe that $V_{\pi^*}(s|T')$ is monotonically non-decreasing and later diverges from $V_{\pi}(s|T')$. This is because the rate of order satisfaction of the *GPU Preferred Supplier* has improved to an extent where the new supplier replaces the *Inventory* as the option of choice in the optimal policy.

All that remains now is to find the boundary points, say $\mathbf{p_{min}}$ and $\mathbf{p_{max}}$, of the region of overlap and prune the range $[\mathbf{p_{min}}, \mathbf{p_{max}}]$ from consideration while calculating the VOC. Formally,

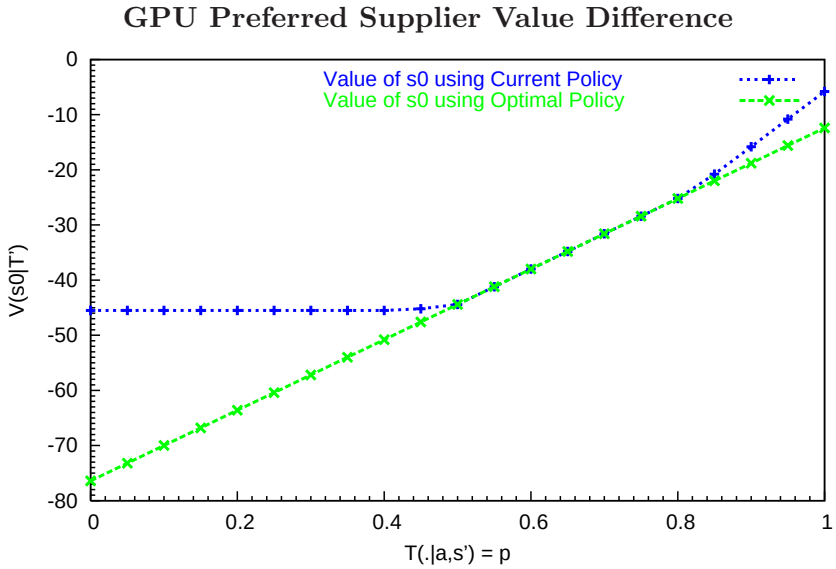


Figure 5.2: The plot of $V_{\pi*}(s|T')$ and $V_{\pi}(s|T')$ for the GPU Preferred Supplier in the start state of the MS XBOX supply chain.

$$\int_{\mathbf{p}} Pr(T'(\cdot|a,s') = \mathbf{p}) \overbrace{[V_{\pi}(s|T') - V_{\pi^*}(s|T')]}^{\Delta V} d\mathbf{p}$$
$$= \int_{\mathbf{0}}^{\mathbf{p_{min}}} Pr(T'(\cdot|a,s') = \mathbf{p}) \Delta V d\mathbf{p} + \int_{\mathbf{p_{max}}}^{1} Pr(T'(\cdot|a,s') = \mathbf{p}) \Delta V d\mathbf{p} +$$
$$\int_{\mathbf{p_{min}}}^{\mathbf{p_{max}}} Pr(T'(\cdot|a,s') = \mathbf{p}) \Delta V d\mathbf{p}$$
$$= \int_{\mathbf{0}}^{\mathbf{p_{min}}} Pr(T'(\cdot|a,s') = \mathbf{p}) \Delta V d\mathbf{p} + \int_{\mathbf{p_{max}}}^{1} Pr(T'(\cdot|a,s') = \mathbf{p}) \Delta V d\mathbf{p}$$

$$(V_{\pi}(s|T') - V_{\pi^*}(s|T') \text{ is } 0 \text{ for } \mathbf{p_{min}} \text{ to } \mathbf{p_{max}})$$

In the next subsection, I present a fast numerical method to compute the boundary points.

5.2.1 Computing the Intersection Points Using Gradient Descent

The intersection points, $\mathbf{p_{min}}$ and $\mathbf{p_{max}}$, may be computed in different ways. We could formulate a quadratic program (QP) that would minimize the difference betIen $V_{\pi^*}(s|T')$ and $V_{\pi}(s|T')$ and return the maximum and minimum values of $\mathbf{p}$, where the difference is

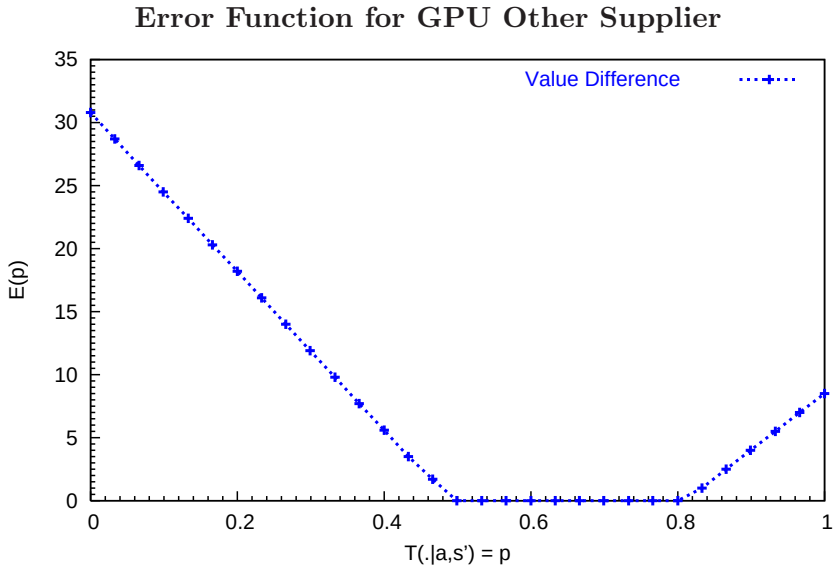


Figure 5.3: The error landscape where the error is $V_{\pi^*}(s|T')$ - $V_{\pi}(s|T')$ for the GPU other supplier. The plateau signifies the region of zero error.

the lowest (= 0). However, not only is the formulation of such a QP complex, but solution methods for general purpose QPs are not yet well-developed.

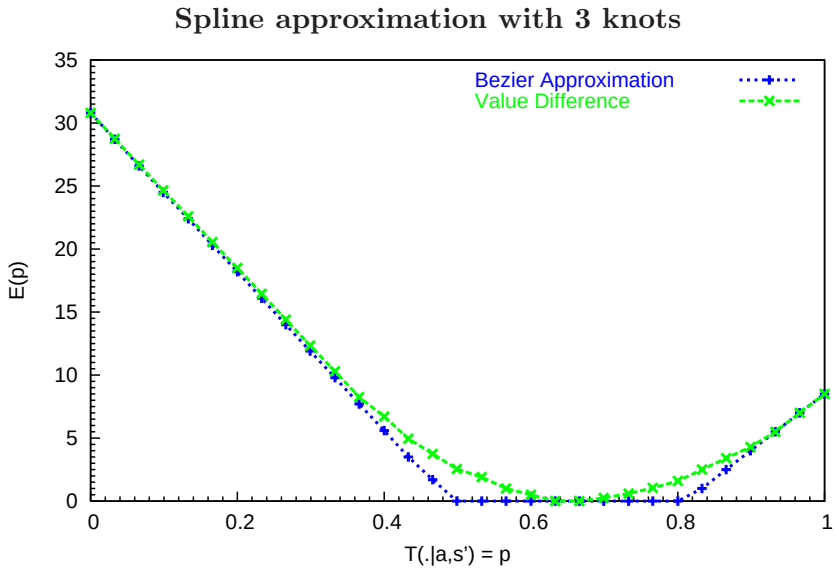
Alternately, we may view the problem of finding the intersection points as that of descending down the error surface until we reach a plateau. The *error* is the difference between $V_{\pi^*}(s|T')$ and $V_{\pi}(s|T')$. The error landscapes for the plots of Figure 5.2 are shown in Figure 5.3. The typical drawback of this approach, called gradient descent [45] – getting stranded on local rather than global minimas – is not a concern here because of the presence of always a single minimum in the landscape.

Within the gradient descent approach, we update the parameter, $\mathbf{p}$, in the following way:

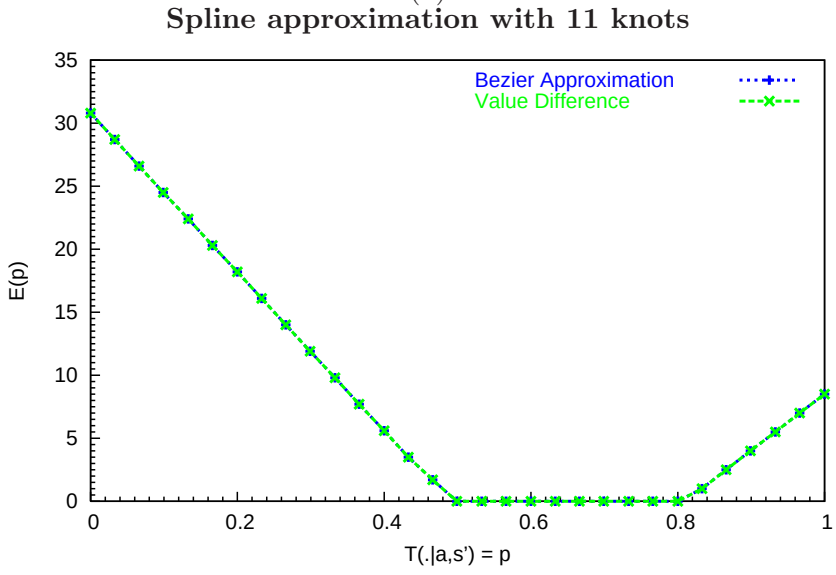
$$\mathbf{p} \leftarrow \mathbf{p} + \Delta\mathbf{p} \;\; ; \;\; \Delta\mathbf{p} = -\eta \frac{\partial E(\mathbf{p})}{\partial \mathbf{p}} \tag{5.1}$$

Here, η is the step size, $0 \leq \eta \leq 1$, the negative sign indicates that we take a step in the direction of the reducing gradient, and the error function, $E(\mathbf{p}) = V_{\pi^*}(s|T') - V_{\pi}(s|T')$, $T'(\cdot|a, s') = \mathbf{p}$. Beginning from an initial value, we continuously update $\mathbf{p}$ until the difference

between the revised and the previous values of $\mathbf{p}$ becomes very small. As evident form Figure 5.3, the initial values will ideally be close to the extremes (i.e. maximum and minimum values in the range of $\mathbf{p}$) and gradient descent will be applied in both directions from the right and left to find the boundary points.



(*a*)



(*b*)

Figure 5.4: The error function for the GPU Other Supplier approximated using splines of (*a*) 3 knots (2 Bezier curves) and (*b*) 11 knots (10 Bezier curves).

Computing the partial derivative of the error function, $\frac{\partial E(\mathbf{p})}{\partial \mathbf{p}}$, is difficult because of the recursive nature of the Bellman equation (Eq. 2.1). To avoid this arduous step, we may

approximate the error surface with a set of basis functions that are relatively simple to differentiate. The particular shape of the error surface and the need for functions that are easily differentiable, suggests that *polynomial splines* [12] may serve as good candidates. A spline is a general piecewise function where the pieces are polynomials and is capable of approximating any shape up to arbitrary accuracy. Because the pieces are polynomials, a spline is relatively simple to differentiate.

Formally, a polynomial spline is defined as, $S : [a, b] \rightarrow \mathbb{R}$, which consists of polynomial pieces, $P_i : [t_i, t_{i+1}) \rightarrow \mathbb{R}$, where: $a = t_0 < t_1 < ... < t_{k-2} < t_{k-1} = b$. The k points, $t_j : j = 0 \ldots k-1$ are referred to as the *knot vector*. Knots represent the border points of pieces of the polynomial.

k equidistant knots are uniformly selected. Selecting the j^{th} knot involves computing, $V_{\pi^*}(s|T' = \mathbf{p_j}) - V_{\pi}(s|T' = \mathbf{p_j})$, where $\mathbf{p_j}$ is the x-coordinate of the knot. Given the knots for the spline, we may now formulate the polynomials at each of the intervals. A popular parametric representation for the polynomials is the *Bèzier* curve of degree d, which may be generalized as follows: Given points, $P_0, P_1, ..., P_d$, the Bèzier curve is:

$$B(\theta) = \sum_{i=0}^{d} \binom{d}{i} P_i (1-\theta)^{n-i} \theta^i \tag{5.2}$$

where P_0 and P_d are the evaluations of consecutive knots, t_j and t_{j+1}, respectively, and θ is the parameter, $0 \leq \theta \leq 1$. For a Bèzier curve of degree d, $d+1$ points are needed. Of course, the more knots selected and the higher the degree of the Bèzier curve, the more accurate the approximation will be.

In Figure 5.4, I show the original error surface of Figure 5.3 as well as the approximation using a polynomial spline where the polynomial pieces are Bèzier curves of degree at most two. I selected 3 equidistant knots (5 points in all) for computing the spline shown in Figure 5.4(a) and 11 knots (21 points) for computing the spline of Figure 5.4(b). The latter spline closely approximates the original error surface, though at the expense of computing a larger knot vector.

Using the x-axis to represent the rate of order satisfaction $\mathbf{p}$ and the y-axis to represent the error $\mathbf{E}(\mathbf{p})$ we may say:

$$\frac{\partial E(\mathbf{p})}{\partial \mathbf{p}} = \frac{\partial y}{\partial x}$$

Given end points P_0^j, where $P_{0,x}^j$ and $P_{0,y}^j$ is the x and y component of point 0 respectively of the j^{th} Bezier curve, and P_2^j and control point P_1^j, the equations for the quadratic Bezier curve are as follows:

$$\begin{aligned} x(\theta) &= (1-\theta)^2 P_{0,x}^j + 2\theta(1-\theta)P_{1,x}^j + \theta^2 P_{2,x}^j \\ y(\theta) &= (1-\theta)^2 P_{0,y}^j + 2\theta(1-\theta)P_{1,y}^j + \theta^2 P_{2,y}^j \end{aligned} \tag{5.3}$$

where $\theta \in [0,1]$ is a parameter. The equation for x may be rewritten as follows:

$$x(\theta) = P_{0,x}^j + (2P_{1,x}^j - 2P_{0,x}^j)\theta + (P_{0,x}^j + P_{2,x}^j - 2P_{1,x}^j)\theta^2$$

Let us focus on the final θ^2 term. Let the interval, i = ($P_{2,x}^j$ - $P_{0,x}^j$) and let points P_0^j, P_1^j and P_2^j be uniformly chosen over x. The term may be rewritten as:

$$2P_{0,x}^j + i - 2(P_{0,x}^j + i/2) = 0$$

we can therefore eliminate the θ^2 term and solve for θ:

$$\theta = -\frac{(P_{0,x}^j - x)}{2P_{1,x}^j - 2P_{0,x}^j}$$

we may now substitute this term for θ into $y(\theta)$ found in Eq. 5.3.

$$\begin{aligned} y = \quad & (1 - (-\frac{(P_{0,x}^j - x)}{2P_{1,x}^j - 2P_{0,x}^j}))^2 P_{0,x}^j + (-\frac{(P_{0,x}^j - x)}{2P_{1,x}^j - 2P_{0,x}^j})^2 P_{0,x}^j \\ & +2(-\frac{(P_{0,x}^j - x)}{2P_{1,x}^j - 2P_{0,x}^j})(1 - (-\frac{(P_{0,x}^j - x)}{2P_{1,x}^j - 2P_{0,x}^j}))P_{1,x}^j \end{aligned}$$

Grouping together terms of x^2, x, and a constant we obtain:

$$\begin{aligned} y \quad = & \frac{P_{0,y}^j + P_{2,y}^j - P_{1,y}^j}{4(P_{1,x}^j)^2 - 8P_{1,x}^j P_{0,x}^j + 4(P_{0,x}^j)^2} x^2 \\ & -(\frac{2P_{0,x}^j(P_{0,y}^j + P_{2,y}^j - 2P_{2,y}^j}{4(P_{1,x}^j)^2 - 8P_{1,x}^j P_{0,x}^j + 4(P_{0,x}^j)^2} - \frac{P_{1,y}^j - P_{0,y}^j}{P_{1,x}^j - P_{0,x}^j})x \\ & +(P_{0,y}^j + \frac{(P_{0,x}^j)^2(P_{0,y}^j + P_{2,y}^j - 2P_{1,y}^j)}{4(P_{1,x}^j)^2 - 8P_{1,x}^j P_{0,x}^j + 4(P_{0,x}^j)^2} - \frac{P_{0,x}^j(P_{1,y}^j - P_{0,y}^j)}{P_{1,x}^j - P_{0,x}^j}) \end{aligned}$$

Finally, we simply take the derivative of y with respect to x:

$$\begin{aligned} \frac{dy}{dx} = \quad & 2\frac{(P_{0,y}^j + P_{2,y}^j - 2P_{1,x}^j)}{4(P_{1,x}^j)^2 - 4P_{1,x}^j P_{0,x}^j + (P_{0,x}^j)^2} x \\ & +(\frac{P_{1,y}^j - P_{0,y}^j}{P_{1,x}^j - P_{0,x}^j} - \frac{2P_{0,x}^j(P_{0,y}^j - P_{2,y}^j - 2P_{1,y}^j}{4(P_{1,x}^j)^2 - 4P_{1,x}^j P_{0,x}^j + (P_{0,x}^j)^2}) \end{aligned}$$

Differentiating the polynomial spline piecewise and utilizing Eq. 5.1, we get the following piecewise rule for updating the parameter $\mathbf{p}$:

$$p \leftarrow p - \eta \begin{cases} 2\frac{(P^1_{0,y} - 2P^1_{1,y} + P^1_{2,y})}{(4(P^1_{1,x})^2 - 8P^1_{1,x}P^2_{0,x} + 4(P^1_{0,x})^2)}p - & t_0 \le p < t_1 \\ (\frac{2P^1_{0,x}(P^1_{0,y} - 2P^1_{1,y} + P^1_{2,y})}{4(P^1_{1,x})^2 - 8P^1_{1,x}P^1_{0,x} + 4(P^1_{0,x})^2} - & \\ \frac{(P^1_{1,y} - P^1_{0,y})}{(P^1_{1,x} - P^1_{0,x})}) & \\ \cdots & \cdots \\ 2\frac{(P^{k-1}_{0,y} - 2P^{k-1}_{1,y} + P^{k-1}_{2,y})}{(4(P^{k-1}_{1,x})^2 - 8P^{k-1}_{1,x}P^{k-1}_{0,x} + 4(P^{k-1}_{0,x})^2)}p - & t_{k-1} \le p < t_k \\ (\frac{2P^{k-1}_{0,x}(P^{k-1}_{0,y} - 2P^{k-1}_{1,y} + P^{k-1}_{2,y})}{4(P^{k-1}_{1,x})^2 - 8P^{k-1}_{1,x}P^{k-1}_{0,x} + 4(P^{k-1}_{0,x})^2} - & \\ \frac{(P^{k-1}_{1,y} - P^{k-1}_{0,y})}{(P^{k-1}_{1,x} - P^{k-1}_{0,x})}) & \end{cases}$$

Beginning from an initial value, the gradient descent rule shown previously will update $\mathbf{p}$ until the difference between the revised and the previous values of $\mathbf{p}$ becomes very small indicating that the boundary of the plateau has been reached. We observe that a large step size, η, may cause the gradient descent to reach the plateau faster, but it may also result in overstepping the boundary region. Consequently, larger η may result in undesirable oscillations.

5.2.2 Performance Evaluation

The average run time of executing and adapting the compositions using Pruned Averaging is given in Figure 5.5. By pruning the effective parameter values as described in Section 5.2.1, we reduced the execution times of the VOC based adaptation by approximately a factor of three. Note that these run times include the time consumed in performing the gradient descent. The speedup was consistent across both the example scenarios.

Problem	Query Random	VOC	VOC with Pruned Averaging
Supply Chain	17s	285s	115s
Patient Transfer	15s	101s	34s

Figure 5.5: A comparison of the average execution times of the MS XBox supply chain and patient transfer compositions. By using pruned averaging, the run time of adapting a composition was cut down by approximately a factor of 3.

5.3 Speeding Up Adaptation Using Expiration Times

As mentioned previously, in order to target a candidate service provider for querying for revised information, computing VOC^* required iterating over all the WSs. For large compositions, there could be several participating WSs, making the process of selection computationally intensive. To address this challenge, I use the insight that service providers are often able to guarantee certain characteristics during execution. Guarantees can be described in a service level agreement (SLA). Here, I exploit a particular guarantee, parameter expiration times, to further reduce VOC's computational overhead.

5.3.1 VOC$^{\mathcal{E}}$: VOC with Expiration Times

We use the insight that service providers are often able to guarantee that their order satisfaction rates and other parameters will remain fixed for some time, t_{exp}, after which they may vary. WS providers may define t_{exp} in a WS-Agreement [17] document (shown later in Section 5.3.4).

Given a way to keep track of which WSs' parameters have expired, we may compute the VOC for only those services whose guarantees have expired and target only among these services for querying. This is because a possible query to the others will return back parameter values that are unchanged from those used in formulating the current composition. Thus such queries will cause no adaptation in the composition, and may be safely ignored.

We assume that, in addition to providing revised parameters, the service providers also give the duration of time for which the parameters are guaranteed to remain unchanged. This duration is called the *expiration time* of the revised information. Let a represent the action of invoking the WS, WS^a, $\mathcal{E}$ be the current set of actions representing the invocations of WSs whose guarantees have expired, we can define the maximum VOC, $\text{VOC}^{\mathcal{E}}$, as:

$$VOC^{\mathcal{E}}(s) = \max_{a \in \mathcal{E}} \; VOC_{T'(\cdot|a,s')}(s) \tag{5.4}$$

where $VOC_{T'(\cdot|a,s')}(s)$ is as defined in Eq. 4.2. Note that $\mathcal{E} \subseteq A$. In the worst case, $\mathcal{E} = A$, and all WSs have expired parameters, in which case, $\text{VOC}^{\mathcal{E}}$ collapses to VOC^* defined in Eq 4.4. In the best case, $\mathcal{E} = \phi$, and none of the WSs' parameters have expired, in which case $\text{VOC}^{\mathcal{E}} = 0$. The challenge then is to correctly maintain the set, $\mathcal{E}$, during the lifecycle of the composition.

5.3.2 Algorithm

In Figure 5.6, the algorithm for generating, executing, and adapting the composition to a volatile environment using $\text{VOC}^{\mathcal{E}}$ is presented. The algorithm takes as input the initial state of the composition, and a policy, π_n^*, obtained by solving the model (Eq. 2.3), which prescribes which WS to invoke from each state of the composition.

As mentioned before, we associate with each WS^a participating in the composition, an expiration time, $t_{exp}^{T(.|a,s')}$, during which the parameters of the service are guaranteed not to change. I begin by checking which of the WSs have expired guarantees (lines 6–11) and updating the set, $\mathcal{E}$, with those that have expired. The next step is to compute $\text{VOC}^{\mathcal{E}}$ (Eq. 5.3.3), which suggests a service provider among the expired set, $\mathcal{E}$, to query for revised information that is expected to bring about most change in the composition.

Notice that a WS might expire while we are computing $\text{VOC}^{\mathcal{E}}$. We must anticipate this and add those services in advance to the set, $\mathcal{E}$, so that they are taken under consideration while computing $\text{VOC}^{\mathcal{E}}$. In line 9, the algorithm invokes the procedure in Figure 5.7, which finds out which WSs among the unexpired ones (denoted by $\bar{\mathcal{E}}$) may expire while computing

Algorithm for generating an adaptive composition using VOC^{ϵ}

Input: s_0 *//initial state,* π_n^* *//optimal policy,* H *//horizon*

1. $\mathcal{E} \leftarrow \phi$ *//Set of expired WSs*
2. $t[1..|A|] \leftarrow 0$ *//Time counter for each WS*
3. $s \leftarrow s_0$, $n \leftarrow H$
4. **while** n > 0
6. **for all** $a \in A$
7. **if** $t[a] > t_{exp}^{T(.|a,s')}$
8. $\mathcal{E} \leftarrow \mathcal{E} \cup WS^a$
9. $\mathcal{E} \leftarrow \mathcal{E} \cup AddExpiredServices(t[a], \mathcal{E})$
10. **end if**
11. **end for**
12. **if** $\text{VOC}^{\mathcal{E}}(s) > QueryCost(T'(.|a^*, s'))$
13. **for all** $a \in A$
14. $t[a] \leftarrow t[a] + t_{VOC^{\mathcal{E}}(s)}$
15. **end for**
16. Query service provider a^* for revised information
17. Obtain $t_{exp}^{a^*}$
18. $t[a^*] \leftarrow 0$
19. **for all** $a \in A/\{a^*\}$
20. $t[a] \leftarrow t[a] + t_{QLag}$
21. **end for**
22. Form the new transition function, T'
23. Calculate policy π_n^* using the new MDP model with T'
24. **for all** $a \in A$
25. $t[a] \leftarrow t[a] + t_{\pi_n^*}$
26. **end for**
27. $\mathcal{E} \leftarrow \mathcal{E}/WS^a$
28. **end if**
29. $a \leftarrow \pi_n^*(s)$
30. Execute WS^a
31. Get response of *action* and construct next state, s'
32. **for all** $a \in A$
33. $t[a] \leftarrow t[a] + t_{Response}$
34. **end for**
35. $s \leftarrow s'$, $n \leftarrow n - 1$
36. **end while**

Figure 5.6: Algorithm for adaptive composition using $\text{VOC}^{\mathcal{E}}$.

Algorithm AddExpiredServices

Input: $t[a], a = 1..|A|$ *//Time counter for each WS,*
$\mathcal{E}$ *//Set of expired WSs*
Output: $\mathcal{E}$

1. *added* $\leftarrow$ *false //Flag*
2. **for all** $a \in \bar{\mathcal{E}}$
3. **if** $t[a] + t_{VOC^{\mathcal{E}}(s)} > t_{exp}^{T(.|a,s')}$
4. $\mathcal{E} \leftarrow \mathcal{E} \cup WS^a$
5. *added* $\leftarrow$ *true*
6. **end if**
7. **end for**
8. **if** *added*
9. AddExpiredServices$(t[a], \mathcal{E})$
10. **else**
11. **return** $\mathcal{E}$
12. **end if**

Figure 5.7: Anticipating Ib services that will expire while computing VOC$^{\mathcal{E}}$.

VOC$^{\mathcal{E}}$ and adds these to the set $\mathcal{E}$. Note that if a WS is added to $\mathcal{E}$, the time taken to compute VOC$^{\mathcal{E}}$ may increase, during which other WSs may expire. We consider this by recursively invoking the procedure until no more WSs are added to the set, $\mathcal{E}$. The time taken to compute VOC$^{\mathcal{E}}$, $t_{VOC^{\mathcal{E}}(s)}$, needs to be anticipated; if $t_{VOC(s)}$ is the time taken to compute the VOC (Eq. 4.2), then $t_{VOC^{\mathcal{E}}(s)} = |\mathcal{E}|t_{VOC(s)}$. Notice that $t_{VOC(s)}$ is fixed and may be obtained *a'priori*.

If VOC$^{\mathcal{E}}$ exceeds the cost of querying the service provider, we query the provider for the new parameters, which form the new T' in the composition model. We also add the time taken to perform the VOC$^{\mathcal{E}}$ calculations to the cumulative time counter associated with each WS (lines 12–16). On querying, in addition to obtaining the possibly revised information, we also obtain the new expiration times for the information. Thus, the counter for the WS that is queried is reset.

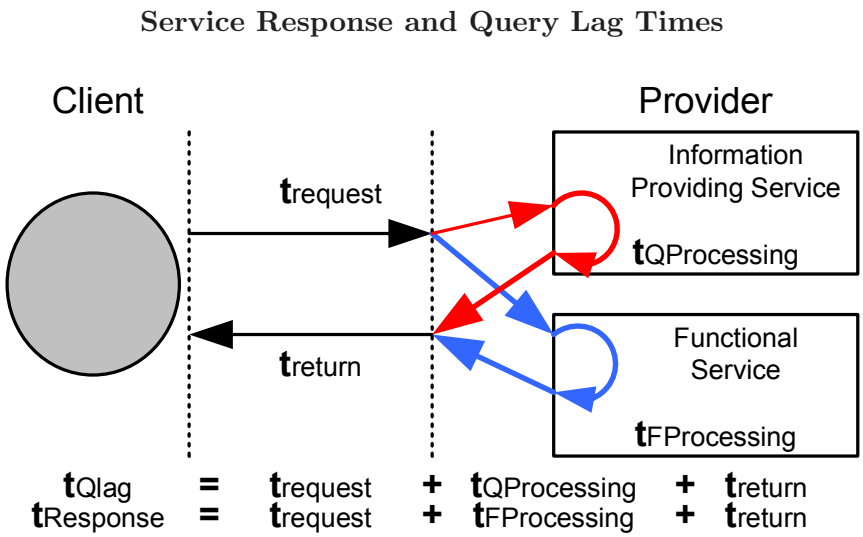


Figure 5.8: Time elapsed in querying for revised information, t_{QLag} (in red), and receiving a response, $t_{Response}$ (in blue), from a service provider.

We observe that querying for information is not a constant time step operation, but must take into account the time taken for the request to reach the provider, the provider's information-providing WS to complete its computations, and for the response to arrive back at the composition. We denote the total time consumed in querying as t_{QLag}, which is depicted in Figure 5.8 [1].

The revised information is integrated into the composition model and a new policy is recomputed to maintain optimality of the composition. However, re-computation of the policy is not always necessary, and runtime changes could be made to the composition. Here, the time counters must be updated again to account for the time taken to recalculate the policy. Finally, the queried WS is removed from $\mathcal{E}$ (lines 22-27). We observe that the times, t_{QLag} and $t_{\pi_n^*}$ could be calculated in *real-time* (online) using timestamps before and after the calculations.

[1]For simplicity, I assume that $t_{Qlag} < t_{exp}$, so as to avoid having revised values expire before they are received. If this condition were not true, it would be best not to query.

Of course, if the query cost exceeds $VOC^{\mathcal{E}}$, then we ignore the previously mentioned steps, and simply invoke the WS that the original policy recommends. Obtaining a response from the invoked WS may not be a constant time operation but may depend on external factors, as shown in Figure 5.8. Let $t_{Response}$ be the time elapsed, then this time is added to all the cumulative time counters (lines 29-34).

5.3.3 Complexity Analysis

I first show that given an identical input, adaptation using $\text{VOC}^{\mathcal{E}}$ results in the same composition as compared to adaptation using VOC^{*} (Eq. 4.4).

Proposition 5.3.1 (Correctness) *Given identical policies and start states, adaptation using $VOC^{\mathcal{E}}$ and VOC^{*} generate identical compositions.*

Proof We begin with the definition of VOC^{*} (Eq. 4.4):

$$\begin{aligned} VOC^{*}(s) &= \max_{a \in A} VOC_{T'(\cdot|a,s')}(s) \\ &= \max_{a \in \mathcal{E} \cup \bar{\mathcal{E}}} VOC_{T'(\cdot|a,s')}(s) \end{aligned}$$

where $\bar{\mathcal{E}}$ is the complement of $\mathcal{E}$ as mentioned previously.

We consider two cases: (i) If the WS with the maximum VOC, selected for querying, has expired parameters, $a^{*} \in \mathcal{E}$, then $\text{VOC}^{*} = \text{VOC}^{\mathcal{E}}$ for every state, and the composition will be adapted identically to when $\text{VOC}^{\mathcal{E}}$ is used. On the other hand, (ii) if the WS parameters associated with the maximum VOC has not expired, then since the revised information is guaranteed to be unchanged, the belief distribution over parameters collapses to a point estimate, $\text{VOC}^{*} = 0$, and the composition remains unchanged. Thus, for both the cases, the resulting composition will be identical to the one generated when $\text{VOC}^{\mathcal{E}}$ is used.

Next, I derive the complexity of the improvement of VOC^{ϵ}.

Proposition 5.3.2 (Improvement of Time Complexity by VOC$^{\mathcal{E}}$) *Let N denote the number of possible values a random variable X can take. The worst-case complexity of adaptation using VOC$^{\mathcal{E}}$, as performed by the algorithm shown in Figure 5.6, is:*

$$\mathcal{O}(H(N^{2|X|}|A|^2|H| + t_{QLag} + t_{Response})).$$

Here, the complexity is quadratic in the number of services ($|A|$). The best-case complexity of adaptation using VOC$^{\mathcal{E}}$ is:

$$\Omega(H(N^{2|X|}|A||H| + t_{Response})),$$

which is linear in $|A|$. However, the tight-bound complexity of adaptation using traditional VOC is:

$$\Theta(H(N^{2|X|}|A|^2|H| + t_{QLag} + t_{Response})).$$

Note that here, the worst case complexity is the same as the best case complexity, both of which are quadratic in $|A|$.

Proof I refer to the algorithm for adaptation using VOC$^{\mathcal{E}}$ shown in Figure 5.6. The outer while loop (line 4) will terminate when the composition has completed (taking at most H steps).

Within the body of the loop we focus on three operations in particular. First, lines 6-11 update the set of expired services, $\mathcal{E}$. Here, a loop iterates over all WSs (ie,$|A|$) effectively having an execution time in the order of $\mathcal{O}(|A|)$. However, each pass of the loop calls the *AddExpiredServices* procedure (Fig 5.7), which terminates when no more services are added to $\mathcal{E}$. In the worst case, this procedure will add one service to $\mathcal{E}$, and then $t_{VOC^{\varepsilon}}$ will increase such that one more service will expire, in which case another service will be added to $\mathcal{E}$; this process is repeated until all the services are added. We may then write the following recurrence for the runtime of this procedure:

$$T(|A|) = \mathcal{O}(|A|) + T(|A| - 1) \tag{5.5}$$

Eq. 5.5 shows that each pass of the loop takes $\mathcal{O}(|A|)$ and, in the worst case, one service will be added to $\mathcal{E}$ for each pass. This recurrence will run in $\mathcal{O}(|A|^2)$ time. Subsequently, lines 6-11 will take $\mathcal{O}(|A|^3)$. Second, line 12 involves a calculation of VOC$^{\mathcal{E}}$, which in the worst case collapses to a VOC* calculation (when all services have expired). VOC*, as mentioned in Eq. 4.4, is the maximum VOC over all services involved in the composition, so $|A|$ VOC calculations are required. Each VOC calculation involves a comparison betIen values produced by the optimal policy and the current policy in the changed environment. With $|X|$ variables having maximum values N, the maximum state space size is $N^{|X|}$. Thus, solving the MDP and finding the optimal policy of a composition takes $\mathcal{O}(N^{2|X|}|A|H)$ time. Similarly, recalculation of the policy in line 23 will also take this time. In total, VOC$^{\mathcal{E}}$ will run in $\mathcal{O}(N^{2|X|}|A|^2H)$ time. Finally, we must also consider t_{QLag} (line 20) and $t_{response}$ (line 33), as these are external to the composition and independent of the VOC calculations.

Thus the total runtime complexity is:

$$\begin{aligned}\mathcal{O}(VOC^{\mathcal{E}}) = \;& \mathcal{O}(|N|^{2|X|}|A|^2H^2) + \mathcal{O}(H \times |A|^3) + \\ & \mathcal{O}(H \times t_{QLag}) + \mathcal{O}(H \times t_{Responsne})\end{aligned}$$

we can eliminate the second term term because $N^{2|X|} >> |A|$. So now we are left with the following complexity:

$$\begin{aligned}\mathcal{O}(VOC^{\mathcal{E}}) = \;& \mathcal{O}(N^{2|X|}|A|^2H^2) + \\ & \mathcal{O}(H \times t_{QLag}) + \mathcal{O}(H \times t_{Responsne})\end{aligned}$$

which may be rewritten as:

$$\mathcal{O}(H(N^{2|X|}|A|^2H + t_{QLag} + t_{Responsne}))$$

However, lines 12-28 are not necessarily executed for each state in the composition. It is possible that a service has not expired, thus $|\mathcal{E}|$ is not equal to $|A|$. In these situations, the intensive computations required at lines 22 and 23 can be ignored. In the best case, none of these computations are required, yielding:

$$\Omega(VOC^{\mathcal{E}}) = \Omega(H(N^{2|X|}|A|H + t_{Responsne}))$$

Note that since none of the services will be queried, the cost of t_{QLag} may also be removed.

Traditional VOC has a worst case scenario equivalent to the worst case of VOC$^{\mathcal{E}}$:

$$\mathcal{O}(VOC^{)} = \mathcal{O}(H(N^{2|X|}|A|^2H + t_{QLag} + t_{Response}))$$

However, using traditional VOC requires all WSs be polled at *every* time step of the composition. That is, lines 12-28 will be executed in every step and $|\mathcal{E}|$ will be equal to $|A|$. Also, the query lag time t_{Qlag} must be added to the complexity. Thus, the best case scenario will resemble the worst case.

$$\Omega(VOC) = \Omega(H(N^{2|X|}|A|^2H + t_{QLag} + t_{Response}))$$

Because the best case and the worst case complexities are the same, I may give the tight-bound complexity of:

$$\Theta(VOC) = \Theta(H(N^{2|X|}|A|^2H + t_{QLag} + t_{Response}))$$

5.3.4 Performance Evaluation

The algorithm described in Figure 5.6 can easily be implemented on the architecture described in Figure 4.3. Within our SOA, we provide internal WSs for solving the MDP model of the composition problem and generating the policy, and computing the VOC$^{\mathcal{E}}$. If the VOC$^{\mathcal{E}}(s)$ exceeds the cost of querying a particular service provider (this cost is also provided as an input), the WS-BPEL flow invokes a special WS whose function is to query the service provider's information-providing WSs for revised information and the new expiration times. This information is used to formulate and solve a new MDP and the output policy is fed back to the WS-BPEL flow. This policy is used by the WS-BPEL flow to invoke the prescribed external WS and the response is used to formulate the next state of the composition. This procedure continues until the steps are exhausted.

The objective of the experimental evaluation is to now show that the average execution time of the composition adapted using VOC$^{\mathcal{E}}$ is less than when the composition is adapted using VOC*, and varies intuitively as the expiration times vary.

```
<wsag:Agreement Name="xs:MS XBox GPU Contract">
   ...
  <wsag:Context>
    <wsag:ServiceProvider>
       xs:GPUPreferredSupplierURI
    </wsag:ServiceProvider>
    <wsag:ExpirationTime>11:59 27 Jan 2009</wsag:ExpirationTime>
    <wsag:TemplateId>...</wsag:TemplateId>
    <wsag:TemplateName>...</wsag:TemplateName>
  </wsag:Context>
  <wsag:Terms>
    <wsag:All>
      <wsag:ServiceDescriptionTerm
        wsag:Name="Rate of GPU Satisfaction"
        wsag:ServiceName="Order GPUs" >
        <job:SatisfactionRate>0.4</job:SatisfactionRate>
      </wsag:ServiceDescriptionTerm>
    ...
    </wsag:All>
  </wsag:Terms>
< /wsag:Agreement>
```

Figure 5.9: A WS-Agreement document showing the agreed upon expiration time and the rate of order satisfaction of a supplier for the XBox supply chain.

I again utilize the MS XBOX supply chain and the clinical patient transfer scenarios (Sections 3.1 and 3.2) for evaluation. For the MSXBOX supply chain example, I queried the suppliers for their current percentage of order satisfaction while in the patient transfer pathway, I queried the secondary caregivers for their current vacancy rates. In addition to the revised information about their services, the providers also guarantee a duration over which the WS parameter values will remain fixed. These distributions may be provided by the service providers using the WS-Agreement [17] specification.

Figure 5.9 shows a part of an agreement between MS and the preferred GPU supplier in the XBox scenario. t_{exp} is defined within the $< ExpirationTime >$ within the $< Context >$ tag. Here, the agreement will expire on January 27, 2009 (for example). Further in the doc-

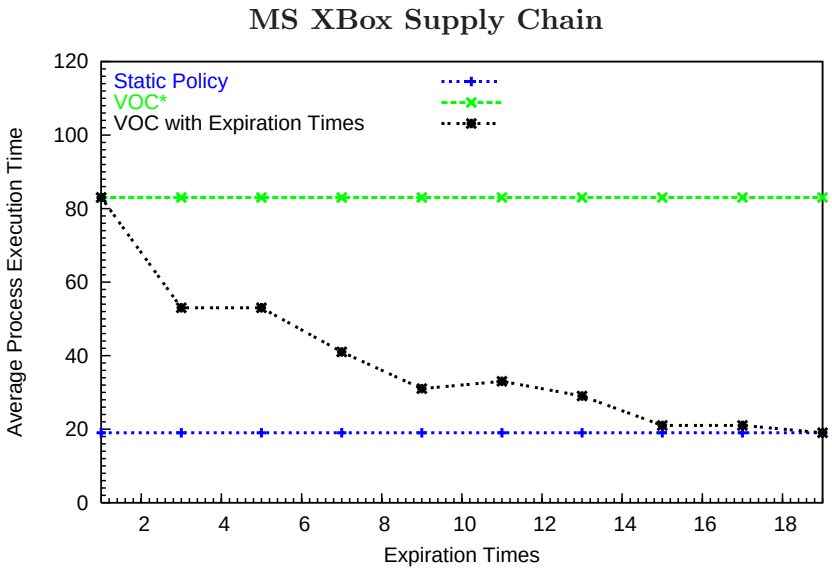
ument, the $< ServiceDescriptionTerm >$ within the $< Terms >$ tag defines the provider's rate of GPU order satisfaction (probability of 0.4). Thus, MS and the contracted GPU manager have agreed that any order of GPUs from MS will be satisfied 40 percent of the time until the agreement is voided on January 27, 2009.

MS XBox Services	**Cost**	t_{QLag}**(s)**	$t_{Response}$**(s)**
GPUPreferredSupplier	30	1	5
GPUOtherSupplier	50	1	4
GPUSpotMarket	80	1	3
ConsolePreferredSupplier	30	1	5
ConsoleOtherSupplier	50	1	4
ConsoleSpotMarket	80	1	3
Patient Transfer Services	**Cost**	t_{QLag}**(s)**	$t_{Response}$**(s)**
InsuranceValidation	10	1	1
PhysicalExam	10	1	2
PreferredSecondaryCaregiver	20	1	3
SecondaryCaregiver2	30	1	3
SecondaryCaregiver3	37	1	3
SecondaryCaregiver4	65	1	3

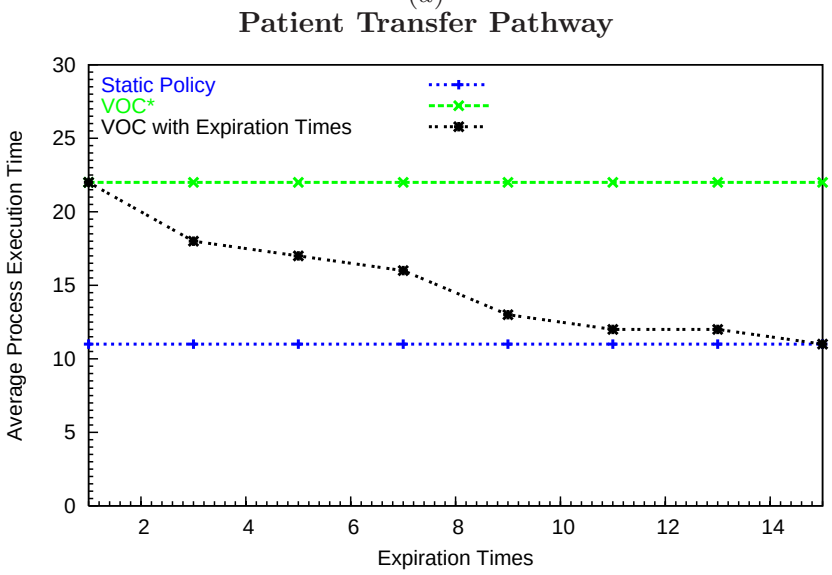
Figure 5.10: Costs, t_{QLag}, and $t_{Response}$ for the WSs in the MS XBox supply chain and Patient Transfer scenarios.

For those services when their information expires, we model the MSXBOX manufacturer's and primary caregiver's beliefs over their possible parameter values, ($Pr\ (T'(\cdot|a, s') = \mathbf{p})$ in Eq. 4.2) using the beta functions given previously in Figure 4.6. For those services whose revised information has not expired, the manufacturer's and caregiver's beliefs could be seen as Dirac-delta functions, with the non-zero probability value fixed at the probability, $\mathbf{p}$, that was provided at the time of query. Thus, for this case, the $VOC(s) = 0$ at any state of the composition.

In order to perform the evaluations, I again simulated a volatile business environment for both the MSXBOX and patient transfer problem domains. The expiration times were upper bound to a large time interval and randomly selected within the bound. The rates of order satisfaction remained fixed until the corresponding expiration times elapsed, after which, on query, new expiration times were randomly selected. The time parameters of the



(a)



(b)

Figure 5.11: (*a*) Average composition execution times (in sec) when using the VOC$^{\mathcal{E}}$ based adaptation, VOC* based adaptation and no adaptation. (*b*) shows analogous results for the patient transfer clinical pathway.

environment, t_{QLag}, and $t_{Response}$, are given in the Figure 5.10. The environment for the patient transfer problem was simulated analogously.

In Figures 5.11(a) and (b), the run times for generating and executing the composition are compared for the MSXBOX and patient transfer scenarios, respectively. We compare the execution time of a composition without any adaptation, with the execution time of a composition adapted using VOC* (with pruning), and the execution time of a composition adapted using VOC$^{\mathcal{E}}$ (Figure 5.6) (also with pruning) [2]. As we increase the expiration times associated with the revised information obtained from the providers, the composition execution time when adapted using VOC$^{\mathcal{E}}$ *decreases*. Notice that it is upper bounded by the execution times of a composition adapted using VOC*, and lower bounded by the run times of a composition without adaptation. This is intuitive because VOC* involves considering *all* participating WSs for querying, while no such computations are carried out in a composition without adaptation. Both these execution times are invariant with respect to the expiration times. The results demonstrate the inverse relationship between expiration times and computational effort expended on adaptation. We also observed that the costs of compositions, in Figs 4.8(a) and (b), do not change supporting the fact that using VOC$^{\mathcal{E}}$ does not affect compositions. This substantiates the intuition that in less volatile environments as formalized by higher expiration times, less adaptation is required to keep the composition optimal.

5.4 Summary

Computing VOC is a computationally intensive operation. However, two methods were proposed to help mitigate the complexity so that the approach becomes more feasible. First, I identified those parameter values that are not expected to cause changes in the composition and may be ignored while computing the VOC. This effectively reduced the range of values needed to compute the VOC. I then exploited service parameter guarantees during which parameters' values remain unchanged, that is, services whose parameter guarantees

[2]Note for clarity of the figure, I chose to omit time plots for the random querying, intermittent querying, and largest difference approaches for querying. Their process times would be slightly greater than that of the static policy if included for comparison

have not expired, need not be considered for querying. Hence, using these techniques allow WSCs to enjoy the cost benefits offered by VOC without paying a steep computation price for adaptation.

Chapter 6

Value of Changed Information For Hierarchical Compositions

In many cases, a WSC may be seen as nested – a higher level WSC may be composed of WSs and lower level WSCs – which induces a natural hierarchy over the composition. For example, the mortgage broker process defined in Section 3.3 In contrast to flat WSCs, a hierarchical decomposition introduces multiple challenges: (*a*) Because only the parameters of the lower level component services are known, we must derive the parameters of the composite service from these, and, (*b*) we must derive a model of volatility for the composite service's parameters from the models of its component services. While previously the usefulness of VOC was demonstrated in the context of simple WSCs, in this Chapter, I focus on utilizing VOC for adapting *hierarchical WSCs*. Section 6.1 gives a brief motivation and definition of hierarchical WSCs. Section 6.2 outlines a way of formulating the stochastic models of volatility for composite services, so that VOC may be applied to hierarchical WSCs. Section 6.3 gives an algorithm for implementing VOC in hierarchical WSCs. Section 6.4 conducts a complexity analysis for finding VOC for hierarchical WSCs. Finally, section 6.5 provides empirical evidence of applying VOC to hierarchical WSCs.

6.1 Hierarchical Web Service Compositions

For manageability, WSCs are often decomposed into a hierarchy. In particular, a WSC may include component services that are themselves WSCs. A WS that is itself implemented as a lower level WSC is often called a *composite* WS.

Multiple approaches for composition exist that exploit a hierarchical decomposition [78, 85]. I slightly modify the approach of Zhao and Doshi [84] and model each level of the

hierarchy using the MDP model of Section 2.1.2. Specifically, the lowest levels of the hierarchy (leaves) are modelled using a MDP containing *primitive* actions, which are invocations of the WSs. Higher levels of the composition problem are modelled using MDPs that contain *abstract* actions, which represent the execution of lower level WSCs. While formulating the lowest level MDPs is straightforward and proceeds as in Section 2.1.2, we must derive the parameters of the composite WS to permit the formulation of the higher level MDP.

Model parameters for abstract action A higher level MDP is so far not well defined because meaningful parameters for the abstract actions in the model are not given. For example, in the mortgage scenario of Figure 3.3, the composite WS, Insurance Information, at level 1 is composed of primitive WSs: Hazard and Flood Insurance Information and one or multiple WSs among CTIC Title Insurance, Delta Title Insurance and TICORE Title Insurance. Transition probabilities associated with the abstract action Collect Insurance Information are not available, but instead must be derived from the transition probabilities associated with the primitive actions.

Zhao and Doshi [84] utilize the correspondence between the high level abstract actions and the corresponding low level primitive actions. Let the abstract action, $\bar{a}$, represent the sequential execution, in some order, of primitive actions, $\{a_1, a_2\}$, of the underlying primitive MDP. Because the order in which the primitive actions a_1 and a_2 are performed is not known from beforehand, there may be multiple ways to achieve the composition – start from the state s_p and reach the state, s_e. Let $s_p \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_e$ be one such path, where s_1 is an intermediate state of the WSC, then, $T(s_1|a_1, s_p) \times T(s_e|a_2, s_1)$ is the probability of following this path, where T is the transition function of the primitive MDP. The required probability, $Pr(s_e|s_p)$, is the sum of the probabilities of following all candidate paths. Analogously, the cost of performing the abstract action is the average of the cost of following each of the possible paths that achieve the composition weighted by the probability of that path.

6.2 Derivation of Aggregated Volatility Models

Analgous to the belief densities described in Section 4.1, we may model the mortgage broker's beliefs over the possible parameters of the WS, ($Pr(T'(\cdot|a, s') = \mathbf{p})$, in Eq. 4.2) using density functions. This time the component Web services take the form of *Gaussian* density functions. Figure 6.1 shows the densities for the Hazard and Flood Insurance Information WS (a) and the Title Insurance WSs (b). Other WSs in the mortgage loan process are assumed to have analogous densities. Means of the densities reveal that the Delta Title Insurance WS tends to be less reliable in satisfying requests than the other title insurers' WSs. Note also that the Hazard and Flood Insurance Information WS is very reliable, having a mean close to 1 and standard deviation relatively small.

Modeling beliefs over the volatile parameters of the composite services is more complex. Section 6.1 mentioned that the transition function of the composite service may be obtained by taking the product of the transition probabilities of the corresponding individual services. We use this in forming beliefs over the volatile parameters of composite services. For example, let us obtain the belief for the Insurance Information composite WS at level 1. The availability of the Insurance Information service is the sum of the products of the availability of Hazard and Flood Insurance service and one or more of the different Title Insurance services. *Therefore, the belief density over the availability of* Insurance Information *will be the summation of functions over the product space: availability of* Hazard and Flood Insurance *WS* $\times$ *availability of a* Title Insurance *service.*

Deriving beliefs for additional flow constructs Composite services such as the Insurance Information WS contain services to be executed in a strictly *sequential* path. Many WSCs, however, exhibit other flow patterns. The derivation of the beliefs must therefore be generalized to accommodate these constructs. Let us consider a composition of two services, $w1$ and $w2$, having beliefs $F(p_1)$ and $F(p_2)$ over their volatile parameters, where $p1, p2 \in [0, 1]$ is the availability of $w1$ and $w2$, respectively. We derive the general belief densities of the composite WS for four commonly used configurations:

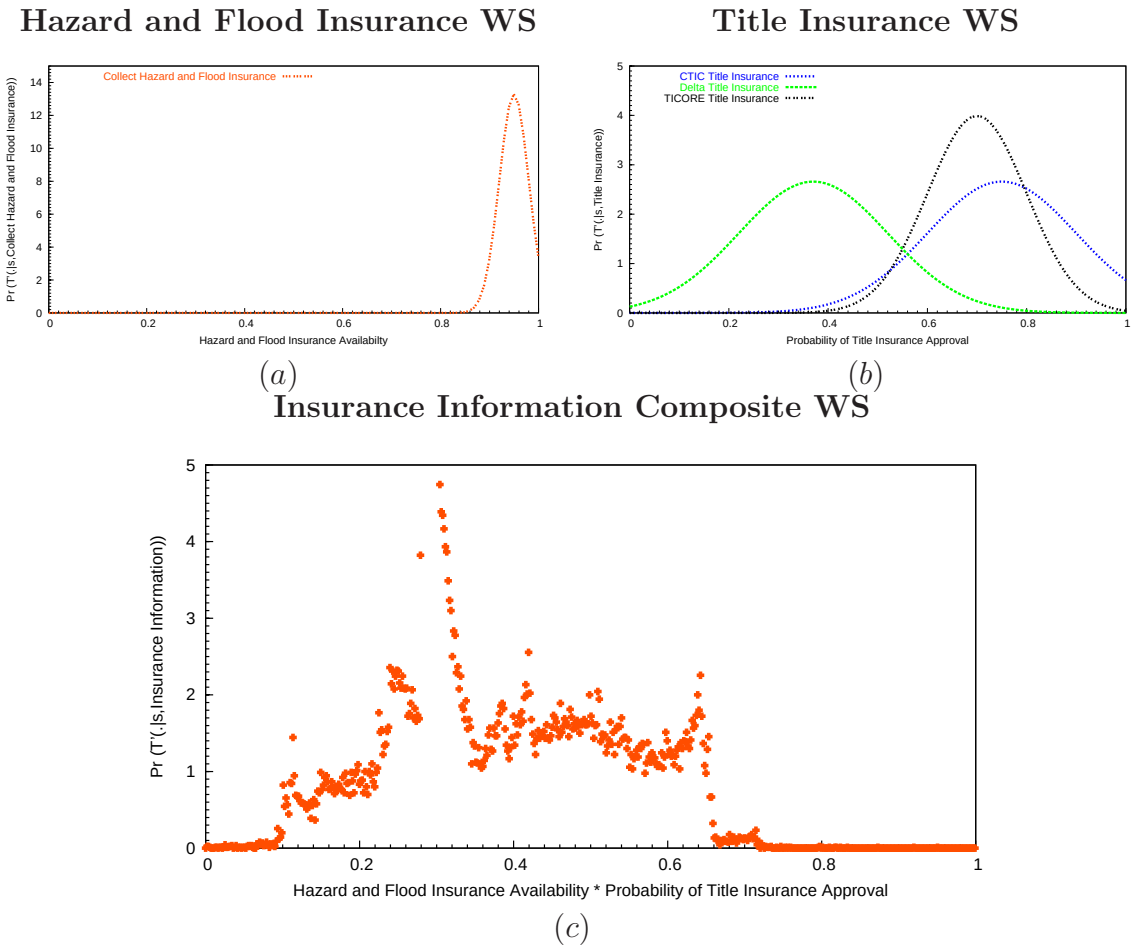


Figure 6.1: (a) Probability density function (pdf) representing the mortgage broker's beliefs over the Hazard and Flood Insurance WSs' probabilities of satisfying requests. (b) Pdfs representing the broker's beliefs over the Title Insurance WSs' probabilities of satisfying requests. (c) The resulting approximate pdf for the composite WS Insurance Information.

- **Sequential flow** Because each of the services w_1 and w_2 in a sequential flow are executed, the composite beliefs will be contingent on both $F(p_1)$ and $F(p_2)$. If executions of the WSs are independent of each other, then the probability of the composite service c containing WSs w_1 and w_2 in sequence, p_c, is the product of the individual WS probabilities: $p_c = \prod_{i=1}^{2} p_i$. Note that because p_c is a product of the individual probabilities, we could view the range of pc as a product space. The density may then be found as

follows:

$$F(p_c) = \int_0^1 F(p_1)F(\frac{p_c}{p_1})\frac{1}{|p_1|}dp_1 \tag{6.1}$$

For simplicity we will use the following notation:

$$F(p_c) = F(p_1) \otimes F(p_2) \tag{6.2}$$

- **Concurrent flow** Analogous to the sequential flow, each of the services in a parallel flow must also be executed. Hence, p is derived analogously to the one for the sequential flow.

- **Conditional flow** Any one of the branches is executed in a conditional flow. For example, let there be two branches that are followed with probabilities, p' and p'' and $(p'+p'' = 1)$. p_c is the weighted sum of the probabilities of the individual WSs: $(p_c = p' \times s_{w1} + p'' \times s_{w2})$. Given the beliefs over the probabilities p_1 and p_2, we may derive the belief density over p_c as:

$$F(p_c) = \frac{1}{p'p''}\int_0^1 F(p_1)F(\frac{p_c - p_1}{p''})dp_1 \tag{6.3}$$

- **Loop** For simplicity, we restrict our analysis to loops that are iterated a fixed number of times, say n [1]. Analogous to the sequential flow, each WS in a loop must be executed n times. Assuming both w_1 and w_2 are both contained in the loop, we may take the product $p'_c = p_1 \times p_2$ n times:

$$F(p_c) = F(p'_c) \otimes F(p'_c) \otimes F('p_c)...ntimes \tag{6.4}$$

I refer the reader to [58] (pg. 141) for more detail on the methods of derivation described above and note that they may be generalized to more services in a straightforward way. Additionally, He et al. [37] investigates how volatility are modelled for dependencies that exist in these workflow patterns.

[1] This precludes loops that conditionally terminate (e.g. while loops). The difficulty is in knowing the number of times the loop will run apriori.

Algorithm for approximate PDF over product space
Input: $\mathcal{N}(\mu_1, \sigma_1)$, $\mathcal{N}(\mu_2, \sigma_2)$

n *//number of samples*
$numBins$ *//number of bins used in histogram, cdf and pdf*
$frequencyCountBins[1..numBins]$ *//freq. count for histogram*
$cdf[1..numBins]$, $pdf[1..numBins]$
Sample densities and tabulate freq. of product of samples
for $i = 1$ **to** n
    Sample $s_1 \sim \mathcal{N}(\mu_1, \sigma_1)$, $s_2 \sim \mathcal{N}(\mu_2, \sigma_2)$
    $p \leftarrow s_1 \times s_2$
    $j \leftarrow$ bin corresponding to p
    Increment $frequencyCountBins[j]$ by 1
end for
Convert $frequencyCountBins$ **histogram to a normalized cdf**
Convert the resulting cdf to a pdf
end algorithm

Figure 6.2: Sampling algorithm for approximating a probability density over a product of two independent random variables with Gaussian distributions.

Sampling Algorithm Although we model the densities over availability of individual WSs as Gaussians (for example Figures 6.1(a) and (b)), the function over the product space is not a Gaussian but rather a modified Bessel function of the second kind [30]. Because generating the Bessel function exactly is complex, we utilize a sampling method to generate the density over the given product space, which converges to the exact as the number of samples approaches infinity. The algorithm for the sampling approximation is given in Figure 6.2. We first sample the individual densities and tabulate the frequencies of the product of the two independent variables. The frequency histogram is converted into a normalized cumulative distribution function (cdf), which may be converted into an approximate probability density function. We show an approximate density obtained by the algorithm in Figure 6.1(c). Densities analogous to this one are summed to obtain the broker's belief over the volatility of the aggregate parameter of the composite WS Insurance Information.

6.3 Algorithm

The algorithm for an adaptive WSC is shown in Figure 6.3. If a^* is a composite WS at level l, we must find the WS at level $l-1$ to query (lines 3-5). This procedure recurses down the nesting level until we select a primitive WS to query. We outline this recursive procedure in Figure 6.4.

Algorithm for adaptive Web service composition – AWSC
Input: $\langle \pi_l^*, \pi_{l-1}^*, \ldots, \pi_0^* \rangle$ *//optimal policies,* s_0 *//initial state* l *//depth,* H *//horizon*

1. $s \leftarrow s_0$
2. $n \leftarrow H$
3. **while** $n > 0$
4. **if** $VOC^*(s) > QueryCost(T'(\cdot|a^*, s'))$
5. **if** a^* is composite
6. $a_k^* \leftarrow findWS^*(l-1, A, s)$
7. Query a_k^* *//primitive WS*
8. $T' \leftarrow UpdateModel$(Level l, Level k)
9. Calculate policy π_l^* using the new MDP with T'
10. $a \leftarrow \pi_l^*(s)$
11. **if** a is composite
12. Recursively call **AWSC** for a at level $l-1$ and π_{l-1}^*
13. **else**
14. Execute primitive Web service a
15. Get response of a and construct next state s'
16. $s \leftarrow s'$
17. $n \leftarrow n-1$
18. **end while**

Figure 6.3: Algorithm for executing and adapting a hierarchical WSC to revised information.

Algorithm for findWS$^*(k, A, s^k)$
Input : k *//level,* A *//action set ,* s^k *//state at level* k

1. $a_k^* \leftarrow \underset{a \in A}{argmax}\ VOC_{T(\cdot|a,s')}(s_k)$
2. **if** a_k^* is a primitive WS **then**
3. return a_k^*
4. **else** // a_k^* is a composite service
5. $s^{k-1} \leftarrow$ initial state of the WSC at level $k-1$
6. return $findWS^*(k-1, A, s^{k-1})$

Figure 6.4: Function $findWS^*$ recursively finds a WS that yields the highest VOC.

After querying a_k^*, where k represents the level at which the queried WS resides, we must formulate a new transition, T', and policy, π^*, for the level k WSC. Subsequently, a new transition function (for the corresponding composite WS) and policy must be computed at all levels up to the top most level, l (lines 6-7). For example, if the CTIC Title Insurance is queried, we reformulate the policy at level 0 given the revised information and recompute the aggregate parameters of the composite WS, Insurance Information, at level 1. We subsequently revise the transition function, T' and resolve π^* at level 1. This recursive procedure is presented in Figure 6.5.

Algorithm for UpdateModel(l, k)
Input : k *//depth,* l *//current level*

1. **if** $l > k$ **then**
2. $T' \leftarrow UpdateModel(l-1,k)$
3. Calculate new policy π_{l-1}^* using the MDP with T'
4. Formulate new T_c for composite WS given T'
6. return T_c
7. **else**
8. Integrate revised information from query to form T'
9. Calculate new policy π_k^* using the MDP with T'
10. return T'

Figure 6.5: Function *UpdateModel* recursively updates the transition probabilities and policies in the hierarchical WSC using the revised information.

6.4 Complexity

The complexity of the algorithm in Figure 6.3 next.

Theorem 6.4.1 *Let N denote the number of possible values a given random variable X can take, $|H|$ denote the number of steps required by the composition to finish, $|A|$ denote the number of services available at each composition level, and d denote the depth of the hierarchical composition. The worst-case complexity of hierarchical adaptation, as performed by the algorithm in Figure 6.3, is:*

$$\mathcal{O}(|H|^{d+1} \cdot (d \cdot (N^{2|X|}|A|^2|H|)) + \mathcal{O}(|H|^{d+1} \cdot (d \cdot N^{2|X|}|A||H| + d \cdot |A|^{|H|})).$$

Proof The outer while loop (line 1) will terminate when the composition has completed (taking at most $|H|$ steps). Within the body of the loop we focus on four operations in particular. First, we find $VOC^*(s)$ as shown in line 2. In Section 5.3, we found that the worst case run time of this operation is $\mathcal{O}(N^{2|X|}|A|^2|H|)$. Second, in line 4, the recursive function $findWS^*$ (see *Figure* 6.4) is triggered when the service a corresponding to $VOC^*(s)$ is a composite service. $findWS^*$ will be recursively called until a primitive service is found. We may write the recurrence definition as follows:

$$T(l) = T(l-1) + \mathcal{O}(1) \tag{6.5}$$

where l is the level of the composition. The first term ($T(l-1)$) defines the recurrence as levels of the hierarchy are traversed and the second term ($\mathcal{O}(1)$) represents the time needed to combine previous recurrences. In the worst case, the procedure will traverse through the entire depth of the composition (d), finding the lowest level composition containing only primitive services. At each level, the algorithm computes $VOC^*(s)$ (line 1), taking $\mathcal{O}(N^{2|X|}|A|^2|H|)$ time. Solving the recurrence will take $\mathcal{O}(d \cdot (N^{2|X|}|A|^2|H|))$ time. Third, in line 6, the composition calls the recursive function $FormNewTransition$ (*Figure* 6.5). We may write this recurrence definition as follows:

$$T(l) = T(l-1) + \mathcal{O}(|A|^{|H|} + N^{2|X|}|A||H|) \tag{6.6}$$

where, again, l is a level of the composition. analogous to Eqn. 6.5, the recurrence (defined by the first term) will traverse to the lowest level composition in the worst case. Each level of the composition, however, requires the additional computations. In line 3 we reformulate T', which requires $|A|^{|H|}$ time to recompute the summation of all possible paths [2]. In line 4, the algorithm requires $N^{2|X|}|A||H|$ to form the new π' (line 4). The cost for these operations is shown in the second term of Eq. 6.6. Using substitution, we find this recurrence to take

[2]Note that the paths and policies of all composition levels are computed before the start of Figure 6.3 and subsequently cached.

$\mathcal{O}((d)\cdot N^{2|X|}|A||H|+d\cdot|A|^{|H|})$ time. Finally, line 10 is a recursive call to the $AWSC$ algorithm itself. In the worst case, the loop will be executed $|H|$ times at each level of the composition. We may write this recurrence as:

$$T(l) = |H| \cdot T(l-1) + \mathcal{O}(1) \tag{6.7}$$

Each iteration of the loop requires all three of the previously described operations. The time needed is the sum of the times needed for these operations:

$$\mathcal{O}(N^{2|X|}|A|^2|H|) + \mathcal{O}((d \cdot (N^{2|X|}|A|^2|H|)) + \mathcal{O}(d \cdot N^{2|X|}|A||H| + d \cdot |A|^{|H|})$$

Note that the first term may be eliminated as it is of smaller order than the second term. Using this sum, solving Eq. 6.7 will give us a worst case runtime:

$$\mathcal{O}(|H|^{d+1} \cdot (d \cdot (N^{2|X|}|A|^2|H|)) + \mathcal{O}(|H|^{d+1} \cdot (d \cdot N^{2|X|}|A||H| + d \cdot |A|^{|H|})) + \mathcal{O}(|H|^{d-1}).$$

The third term may be eliminated because

$$|H|^{d+1} \cdot (d \cdot (N^{2|X|}|A|^2|H|)) >> |H|^{d-1}.$$

. So now we are left with the final worst case runtime:

$$\mathcal{O}(|H|^{d+1} \cdot (d \cdot (N^{2|X|}|A|^2|H|)) + \mathcal{O}(|H|^{d+1} \cdot (d \cdot N^{2|X|}|A||H| + d \cdot |A|^{|H|})).$$

Theorem 6.4.1 reinforces the intuition that the complexity of the hierarchical WSC algorithm will grow exponentially as the number of levels increases. Note that we may apply the pruned averaging and VOC^{ϵ} techniques developed in Chapter 5 for mitigating some of the complexities.

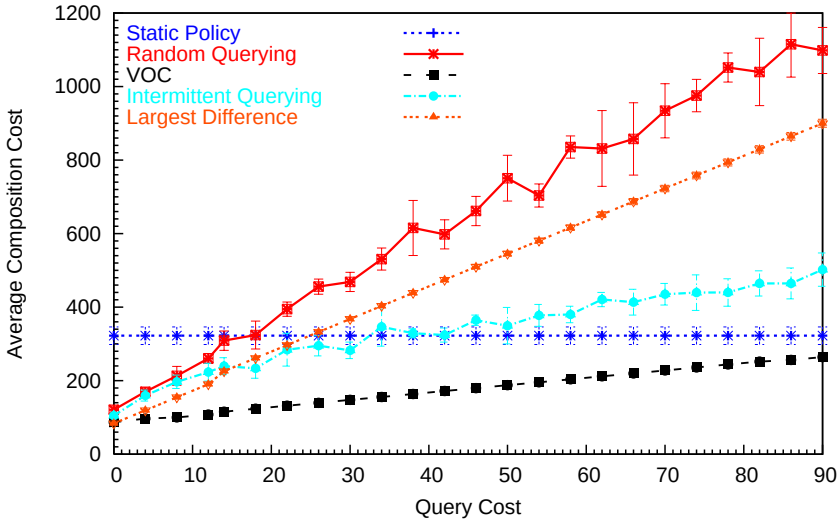
6.5 Performance Evaluation

We utilized the mortgage loan processing scenario (Section 3.3) for our evaluations. We simulated querying the different WSs for their current percentage of request satisfaction (availability). Again, we model the mortgage broker's beliefs over the volatile parameter

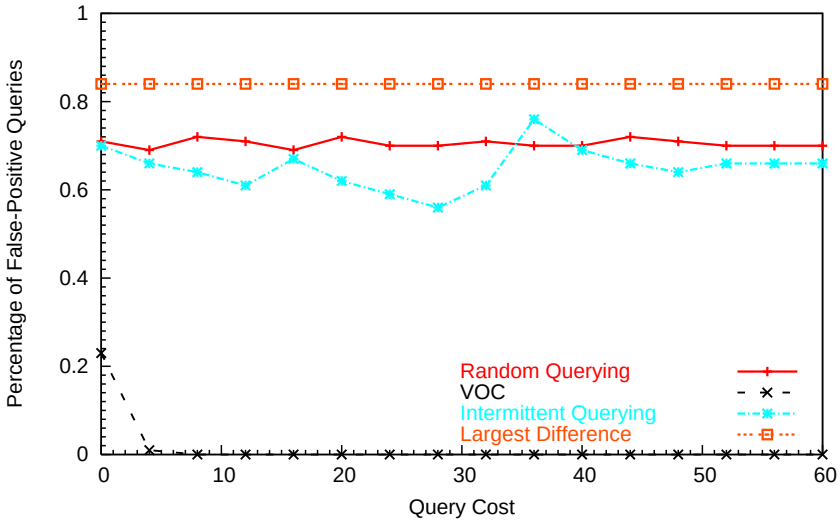
of the individual WSs, ($Pr(T'(\cdot|a, s') = \mathbf{p}$) in Eq. 4.2) using the *Gaussian* density functions shown in Figure 6.1. For composite WSs, we derive the densities over the aggregated parameters as shown in Section 6.2.

In Figure 6.6(a), we compare the VOC-driven selective querying with the four other strategies (introduced in Section 4.4) with respect to the average cost incurred from the execution of the adapted hierarchical WSCs, as the cost of querying the WSs for information is increased. The experiment consisted of running a trial of 500 independent instances of each composition within a simulated volatile environment, where the queried parameters of the services were distributed according to the corresponding density plots (see Figure 6.1). We ensured that the compositions using the different strategies received similar responses from the services.

Intuitively, we see the same trends for each of the querying approaches that we encountered in Section 4.4. As we increase the cost of querying, the VOC based approach performs less queries and adapts the WSC less. For large query costs, its performance is similar to using a WSC with an unchanging policy. For smaller query costs, a VOC based approach will query frequently, though not as much as a strategy that always queries some provider, such as *random query*. As we increase the query costs, the VOC will allow a query for revised information only if its value exceeds the cost. Though *intermittent querying* naively seeks to emulate this behavior, it performs worse because it does not utilize the value of a potential change in the composition in deciding when to query. We note that the *largest difference* approach performs well for lower query costs, though worse than the VOC based approach. This is because the service exhibiting the largest difference from the mean in its parameter value is often the one that brings about the largest change in the composition. However, this is not always the case – for example, a large change in the parameter of a mandatory service, such as the Credit Check service in the mortgage process, does not affect the composition though the approach will query it and incur the query cost. In summary, a WSC that is



(a)



(b)

Figure 6.6: Comparisons of the VOC based adaptive WSC with the static policy and other querying approaches for the mortgage loan acquisition. (a) We measure the average cost. Lower cost indicates better performance. (b) We measure the percentage of false-positive queries. Notice that VOC results in a WSC that is most cost efficient among all strategies, in part, because it issues a much lower percentage of queries that turn out to be false positives.

adapted using VOC incurs less average cost because only significant changes to the WSC are carried out while simultaneously avoiding frequent costly queries.

While it is impossible to guarantee *a'priori* that all issued queries will result in changes in the WSC, we measure the percentage of queries that do not change the WSC – we refer to such queries as false positives. As shown in Figure 6.6(b), VOC based querying results in significantly less false positives compared to other strategies. The number drops to zero as the query cost increases because a query is issued only if the VOC exceeds the cost of querying. Comparative approaches are independent of the query costs. Notice that randomly querying results in approximately 75% of the queries being false positives. We observe that the reduced percentage of false positive queries is responsible, in part, for the reduced cost of adapting WSCs using VOC.

Query strategies	**Time (ms)**		
	QC = 0	**QC = 40**	**QC = 80**
Static Policy	609 ± 50	609 ± 50	613 ± 47
Random query	759 ± 72	755 ± 66	759 ± 72
VOC	2112 ± 196	1800 ± 136	1650 ± 116
Intermittent	653 ± 4	540 ± 20	470 ± 20
Largest difference	535 ± 8	535 ± 12	541 ± 11

Figure 6.7: Running times of the various querying strategies for different query costs.

As evidenced by the complexity discussed in Section 6.4, adapting hierarchical WSCs using VOC comes at a computational price. In Figure 6.7, the run times for each of the comparative approaches is given. To be realistic, a short lag time (5 ms) in receiving the query response from the WSs is included in the time costs. As Figure 6.7, an adaptive WSC that uses VOC runs two to three times slower than a WSC that does not adapt such as the static policy. However, the difference is less when other strategies are utilized. The additional runtime of the VOC is primarily due to the computations required for Eq. 4.2. Notice, however, that as the query cost increases, the time required by the VOC based approach decreases, because it issues less queries.

6.6 Summary

While previously only applied toward adapting simple flat WSCs, we found that we may extend the applicability of VOC to hierarchical (and other complex) WSCs – which is significant because real-world WSCs often tend to have a hierarchy. We were able to obtain beliefs over volatility of composite WS parameters, and which of the component WSs to invoke if a composite WS is found to potentially cause the most expected change in the composition. As a result, when the VOC-based querying method was applied, it was found to be a more effective approach than other state-of-the-art methods.

Chapter 7

Value of Changed Information For Risk-Sensitive Compositions

Traditional VOC aims to adapt compositions in a rational - *risk neutral* - manner. Other risk preferences, however, often strongly influence how modern organizations conduct business and make crucial decisions. Consequently, risk preferences play a pivotal role in determining the goals of its many different business processes. This was demonstrated in a recent survey conducted by Corner and Corner [23], which found that more than a quarter of all business processes show some sensitivity to risk. For example, an organization may be conservative by nature, willing to sacrifice some cost in exchange for more stability and reduced risk of incurring greater costs in the future. In contrast to such *risk aversion*, *risk-seeking* behavior involves making decisions that could yield large gains at the risk of incurring heavier losses. Clearly, a comprehensive approach to composition and adaptation should allow for considerations of risk preferences, as distinct preferences toward risk could significantly affect which WSs are selected in the composition and how the composition is adapted.

In this chapter, I investigate the influence that risk preferences have on generating optimal compositions and subsequently generalize VOC toward modeling risk preferences in deciding which Web service to query for revised information, in order to adapt optimally.

Section 7.1 motivates the need for the consideration of risk preferences. Section 7.2 demonstrates how risk preferences are modelled in WSCs. Section 7.3 formally defines risk-sensitive VOC. Finally, Section 7.4 provides preliminary empirical evidence of the impact of risk preferences on adaptation.

7.1 Motivation

As the inclusion of risk preferences (and their impact) in formulating optimal, adaptive compositions are not immediately straightforward, let us reinvestigate the manufacturers supply chain process described in Chapter 1 (see Figure 1.1). The activities involved in parts procurement with risk preferences is given in Figure 7.1.

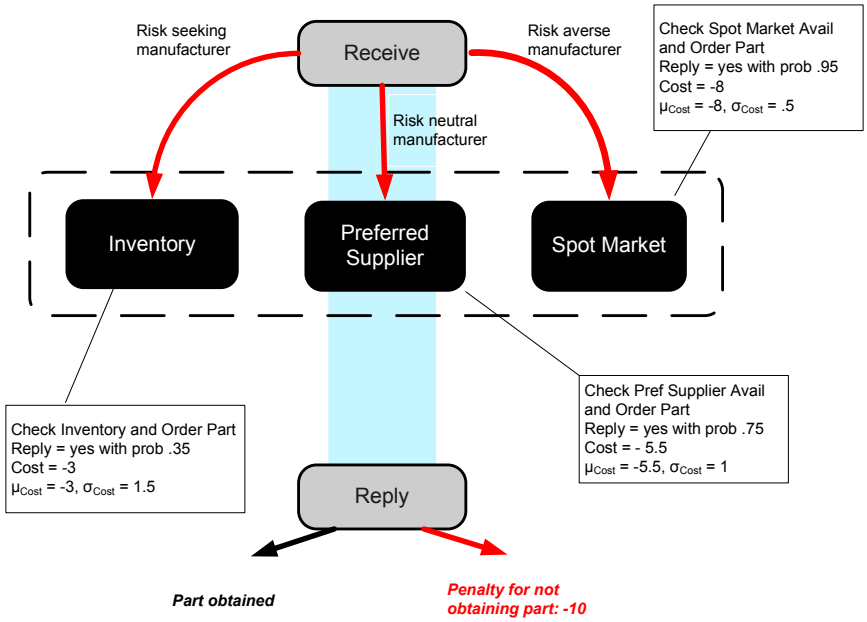


Figure 7.1: The manufacturer's supply chain with risk preferences considered.

Again, the manufacturer must optimally decide between different service providers from whom to obtain specific parts. The first option is to obtain the parts from its own inventory (shown as Inventory in Figure 7.1), an inexpensive option that would allow the manufacturer to acquire the part quickly and cheaply. The manufacturer has limited storage available, however, making this method of obtaining the part unreliable. The manufacturer may also choose to obtain the parts directly from its preferred supplier. The part is more expensive to obtain from the supplier, but its availability is significantly better than from the inventory. Finally, the manufacturer may rely on the spot market, which almost certainly guarantees that the part will be obtained, but is more expensive than the previous two options. If the manufacturer is unable to complete the parts procurement task (i.e., a service is used

that is not able to satisfy the order for the parts), a penalty is incurred in addition to the cost of invoking the WS. This penalty is representative of the recovery costs needed to heal the process (manufacturing halts, process rollbacks, compensations, etc.). We also indicate example QoS properties of each of the available services.

A rational (risk-neutral) manufacturer would optimize the composition by selecting the Web service that maximizes the mathematical expectation of total cost given the probability of obtaining the parts using the WS. For the quality of service (QoS) parameters given in Figure 7.1, a rational manufacturer would elect to use the Preferred Supplier. However, pragmatic manufacturers are not always rational. A manufacturer that is averse to risk would opt for a more sure bet - a reliable service whose probability of meeting the order is thought to be high - despite the potentially higher cost. It therefore seems intuitive that a risk-averse manufacturer would likely select the Spot Market due to almost guaranteed availability of the parts. On the other hand, a risk-seeker would likely bet on a WS that is least expensive despite its lower reliability, such as the Inventory.

An adaptive manufacturer would seek to modify its choice based on the updated information about the component Web services. In this regard, a mean (μ) and standard deviation (σ) that guides the Gaussian distribution of the volatile cost of each participating service is given. Per domain knowledge, the inventory is likely to be the most volatile (i.e. a higher standard deviation) while the spot market to be least volatile.

7.2 Risk-Sensitive Web Service Compositions

The traditional business analysis cycle views risk preferences as an important criteria for designing business processes [38]. Although rational process design stipulates risk indifference (often called *risk-neutrality*), pragmatic composition design often involves either implicit or explicit considerations of risk preferences. For example, surveys [23] have found that 27% of all business processes are designed with some sensitivity to risk. Typically, risk considerations are predominant in processes involving high-stakes decisions, or those involving large sums of

money or resources, in order to either avoid disastrous consequences or obtain huge financial gains [41]. A process designer may be *risk-averse* - it is willing to incur some cost in exchange for more reliability and reduced risk of incurring greater costs in future. Some designers may have opposite preferences and are *risk-seeking* - they make decisions that could yield large possible gains, at the risk of sustaining heavy losses.

7.2.1 Refining the WSC Model

In order to introduce exponential utility functions into the model for composition, I slightly refine the model discussed in Section 2.1.2.

Again, the model of the Web service composition, WSC, is a MDP using a sextuplet:

$$WSC = (S, A, T, R, H, s_0)$$

where $S = \Pi_{i=1}^{n} X^i$, S is the set of all possible states factored into a set, X, of n variables, $X = \{X^1, X^2, \ldots, X^n\}$; A is the set of all possible actions; T is a transition function, $T : S \times A \rightarrow \Delta(S)$, which specifies the probability distribution over the next states given the current state and action; R is a reward function, $R : S \times A \rightarrow \mathbb{R}$, which specifies the reward obtained for performing each action from each state; H is the period of consideration over which the plan must be optimal, also known as the horizon, $0 < H \leq \infty$ [1]; and s_0 is the starting state of the process.

The Bellman equation (previously defined in Eq. 2.1 can be rewritten as follows:

$$V^n(s) = \begin{cases} 0 & s \in \mathbb{G} \\ \underset{a \in A}{max} \ [R(s,a) + \sum_{s' \in S} T(s'|a,s) V^{n-1}(s')] & s \notin \mathbb{G} \end{cases} \tag{7.1}$$

where $V^n(s)$ quantifies the maximum long-term expected reward of reaching each state s with n actions remaining to be performed, and $\mathbb{G}$ represents the set of goal states, indicating

[1] For clarity and simplification of computation, we set horizon H to 1 in this paper, indicating that the MDP will make *greedy* decisions. We note that we may generalize WSC over longer horizons, $H > 1$, in a straightforward manner.

that the process has completed sucessfully [2]. As $H = 1$, Eq. 7.1 reduces to:

$$V^1(s) = \begin{cases} 0 & s \in \mathbb{G} \\ \max\limits_{a \in A} \ [R(s,a) + \sum\limits_{s' \in S} T(s'|a,s)V^0(s)] & s \notin \mathbb{G} \end{cases} \tag{7.2}$$

where $V^0(s') = 0, \forall\ s \in \mathbb{G}$, and $V^0(s') \leq 0, \forall\ s \notin \mathbb{G}$.

Because the reward is negative (cost), Eq. 7.2 implies that the expected value of a non-goal state is less than or equal to the expected value of a goal state. In our example, $V^0(s') \leq 0$ for non-goal states represents the penalty of not procuring the desired parts from the selected supplier. Note that this concept combined with Eq. 7.2 implies that:

$$V(s), \forall s \in \mathbb{G} \geq V(s), \forall s \notin \mathbb{G}. \tag{7.3}$$

Once we know the expected reward associated with each state, the optimal action for each state is the one which results in the maximum expected reward.

$$\pi^*(s) = \underset{a \in A}{argmax} \ [R(s,a) + \sum_{s \in S} P(s|a,s')V^0(s')] \tag{7.4}$$

In Eq. 7.4, π^* is the optimal policy which is simply a mapping from states to actions, $\pi^* : S \rightarrow A$ (analogous to Eq. 2.3). We may easily apply the algorithm in Figure 4.2 to perform the composition.

7.2.2 Traditional Approaches for Compositions with Risk Preferences

There are many ways to model risk preferences in applications. Consequently, risk assessment has been a key research area in economic and business enterprise research communities [46].

Classical approaches to risk management are mostly *qualitative* – human specialists oversee a process and identify potential hazards that could upset its functionality. After the risks are identified, they are addressed in a systematic manner. While these techniques may be applied to SOA-driven applications, they are mostly out of scope with the work in this thesis.

[2] Note that $V(s) \notin \mathbb{G}$ indicates the penalty incurred by the process *not completing* successfully

Others have attempted to represent risk attitude *quantitatively*. Kokash and D'Andrea [40] use traditional risk management strategies to derive contingency plans (such as quality of service re-negotiation or adopting other component services) when the risk of using a composition is found to be high. They operate with the notions of threats (danger sources), probabilities of threats, and their quantifiable impact on the provider of the composition (monetary losses, time losses, breach of reputation, etc). These threats are juxtaposed against the possible gains of the composition. Decisions are made accordingly as to whether to utilize a contingency plan of composition based on these comparisons. Wiesemann et al. [72] incorporate the average Value at Risk (AVaR) measure, widely used in economic studies, into the decision making of a WSC. They introduce risk-preferences using the β-AVaR metric, which is defined as the mean value of the (1-β) worst losses sustained by making a particular decision. $\beta \in [0, 1]$ represents the degree a decision maker considers the worst case loss of a particular decision. When β is 0, the decision maker is risk-neutral. As β increases it becomes more pessimistic, and thus, risk-averse. The β-AVaR metric is introduced to their value maximization equations and decisions are based on the newly constructed equations.

Another common way of modeling risk preferences is by adjusting the *utility function* that maps the actual expected reward to the subjects utility [70, 59]. A utility function, $U : w \rightarrow \mathbb{R}$, quantifies the level of the designer's "satisfaction" based on some current level of wealth, w. Avila-Godoy [11] derived straightforward methods to compute value functions for process designers that utilize exponential utility functions to model their risk attitudes. Liu [41] extended this line of work so that value functions may be computed using more general (i.e. non-exponential) risk-aware utility functions. In this thesis, I borrow from these approaches and utilize exponential utility functions in risk-sensitive Web service compositions.

7.2.3 Exponential Utility Functions

Process designers with risk-neutral preferences utilize a linear utility function, $U(w) = Bw + C$, where B and C are constants, indicating a linear relationship between an increase in

wealth and degree of satisfaction. Typically, utility functions that model risk assume an *exponential* form [23]. Let the risk factor, γ, denote the degree of risk-awareness of a designer. One form of the exponential utility function with respect to wealth, $U_{exp}(w)$, is as follows [41]:

$$U_{exp}(w) = \begin{cases} \gamma^w, & \gamma > 1 \\ -\gamma^w, & 0 < \gamma < 1 \end{cases} \tag{7.5}$$

If $0 < \gamma < 1$, the utility function is concave, indicating that the designer is *risk-averse*. Conversely, if $\gamma > 1$, the utility function is convex, indicating that the designer is *risk-seeking*.

Using the revised model in Section 7.2.1, risk-averse utility (i.e $0 < \gamma < 1$) may be modelled as $U(s) = -\gamma^{U^1(s)}$ and risk-seeking utility (i.e $\gamma > 1$) modelled as $U(s) = \gamma^{U^1(s)}$. Smaller values of γ when $\gamma < 1$ signify greater risk aversion and conversely, larger values of γ when $\gamma > 1$ signify greater risk.

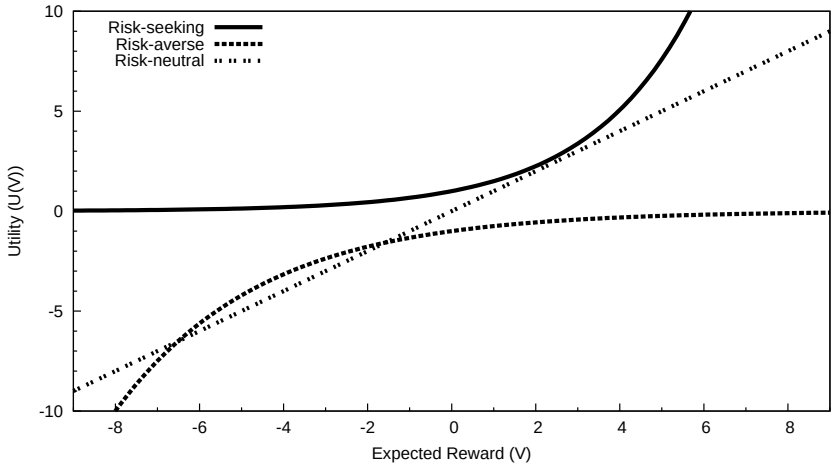


Figure 7.2: Utility functions for different risk preferences. Risk aversion is modelled using a concave function while risk seeking is represented by a convex utility function. The utility function for risk neutrality is linear.

Risk aversion is associated with a large drop in utility for low reward (high cost) while risk-seeking behavior is thought to associate a large increase in utility for positive expected reward. On the other hand, risk neutrality involves considering the expected reward as is. Example utility functions for the three distinct risk preferences is demonstrated in Figure 7.2, where $\gamma = 1.5$ for the risk-seeking function and $\gamma = 0.75$ for the risk-averse utility function.

7.2.4 Impact of Risk Preferences on Composition

Avila-Godoy [11] and Liu [41] show that we may incorporate exponential utility functions with respect to w into an MDP by setting the reward function $R(s,a)$, as an exponent of γ. The value function in Eq. 7.2 may be rewritten as follows:

$$U^1(s) = \begin{cases} \iota & s \in \mathbb{G} \\ \underset{a \in A}{max} \ [\sum\limits_{s' \in S} \gamma^{R(s,a)} T(s'|a,s) U^0(s')] & s \notin \mathbb{G} \end{cases} \tag{7.6}$$

$\mathbb{G}$ are the goal states, $U^0(s') = \iota$, $\forall\ s \in \mathbb{G}$, and $U^0(s') \leq \iota$, $\forall\ s \notin \mathbb{G}$. The value ι is derived from applying the utility function to the corresponding $U^0(s)\ \forall\ s \in \mathbb{G} = 0$ for risk-neutral preferences defined in Eq. 7.2. If the process designer is risk-seeking, $U^0(s)$, $\forall\ s \in \mathbb{G} = \gamma^w = \gamma^0 = 1$. If the process designer is risk-averse, $U^0(s)$, $\forall\ s \in \mathbb{G} = \text{-}\gamma^w = \text{-}\gamma^0 = -1$. Thus, $U^0(s') = \iota$ for all goal states and $U^0(s') \leq \iota$ for all non-goal states. Note that the utility of non-goal states continues to be less than that of goal states.

Considerations of risk could impact the utility of the different states of the WSC to the designer, and potentially which WS invocations are optimal at different states. This implies that the optimal policy, π^*, may be different as well leading to possibly distinct WSCs for different utility functions.

For example, Figure 7.1 illustrates how the traditional, risk-neutral manufacturer selects the optimal service for composition for the scenario in Figure 7.1. The first two columns represent the properties of the supplier WSs: $R(s, a_{WS})$ is the reward of invoking the WS (a negative value representing the cost of invocation) and Av_{WS} is the WS's current rate of order satisfaction. Av_{WS} is used to evaluate the transition funciton, $\sum\limits_{s' \in S} T(s'|a,s) U^0(s)$ (column 3), which determine the long term reward, heavily dependant upon whether the part is obtained (reward of $U^0(s) = 0$) or the penalty for not obtaining the part (reward of $U^0(s) = -10$) is incurred. For example, a composition invoking the Inventory service will satisfy the order 35% of the time, while incurring the penalty 65% of the time, leading to a long-term expected reward value of $((.35) \times (0) + (.65) \times (-10) = -6.5)$. The sum of this long-term expected reward and $R(s, a_{WS})$ yield the expected reward for invoking the WS

WS	**R(s,a)**	Av_{WS}	$\sum_{s' \in S} T(s'\|a,s)U^0(s)$	$U^1_{WS}(s)$
Inventory	-3	.35	$(.35) \times (0) + (.65) \times (-10) = -6.5$	-9.5
Preferred Supplier	**-5.5**	**.75**	$(.75) \times (0) + (.25) \times (-10) = -2.5$	**-8**
Spot Market	-8	.95	$(.95) \times (0) + (.05) \times (-10) = -.5$	-8.5

Table 7.1: Value functions for the different WSs in the risk-neutral manufacturer's composition. The Preferred Supplier is regarded as the optimal service to invoke because the manufacturer's expected utility (rightmost column) of using the Preferred Supplier is greater than the expected utility of using the Inventory and Spot Market WSs.

Service	**U(R(s,a)) =** $\gamma^{R(s,a)}$	Av_{WS}	$\sum_{s' \in S} T(s'\|a,s)U^0(s)$	$U^1_{WS}(s)$
Inventory	$(1.5)^{-3} = .296$	**.35**	$(.35) * (1) + (.65) * (.017) = .361$	**.107**
Preferred Supplier	$(1.5)^{-5.5} = .108$	.75	$(.75) * (1) + (.25) * (.017) = .754$	.081
Spot Market	$(1.5)^{-8} = .039$	.95	$(.95) * (1) + (.05) * (.017) = .951$	.037

Table 7.2: Value functions for the different WSs in the risk-seeking manufacturer's composition. Note that the Inventory is regarded as the optimal service to invoke because it has a larger expected utility than the Preferred Supplier and Spot Market WSs.

(last column). The utility of invoking the Preferred Supplier ($U^1_{PS}(s) = -8$) exceeds the other two options ($U^1_{Inv}(s) = -9.5$ and $U^1_{SM}(s) = -8.5$). The Preferred Supplier, therefore, is viewed as the rational selection for composition.

Risk-sensitive manufacturers will view these values using their corresponding utility functions, leading to a different selection of WSs. Table 7.2 demonstrates how our risk-seeking manufacturer ($\gamma = 1.5$) would select its service for composition. Because the risk-seeking manufacturer's utility function is applied, there is a noticeable change in the values. There are the noticeable differences in utilities of the immediate reward, now $U(R(s,a)) = \gamma^{R(s,a)}$, and the utilities for obtaining the part ($U^0(s) = 1$) and penalty for not obtaining the part ($U^0(s) = .107$). In particular, note that the penalty is viewed as having a smaller impact

than that of the risk-neutral manufacturer. This would imply that the emphasis for reaching the goal (i.e. satisfying the order request) is less than that of a risk-neutral manufacturer. The utility of using the Inventory is greater than the utility of using the other two WSs and is subsequently selected for composition. We may analogously apply the same methodology for the risk-averse manufacturer, which will select the Spot Market.

7.3 Risk-Sensitive VOC

Traditionally, VOC was only used in WSCs indifferent to risk. Specifically, the terms $V_{\pi^*}(s|R')$ and $V_{\pi}(s|R')$ in Eq. 4.2 utilized the risk-neutral utility function defined in Eq. 7.2 in computing the VOC. In order to accomodate a composition's risk sensitivities, we apply the value function given in Eq. 7.6 in computing VOC.

Formally, we may generalize VOC to include considerations of risk preferences by rewriting Eq. 7.1 in the following way:

$$\mathcal{VOC}_{R'(a)}(s) = \int_{\mathbf{r}} Pr(R'(a) = \mathbf{r})[U_{\pi^*}(s|R') - U_{\pi}(s|R')]d\mathbf{r} \tag{7.7}$$

where $\mathcal{VOC}$ denotes the generalized version of the traditional VOC, $U_{\pi^*}(s|R')$ denotes the risk-sensitive utility of the state s given the optimal policy in the context of revised information and $U_{\pi}(s|R')$ denotes the risk-sensitive utility of s given the original policy in the context of revised information. The utility function, U, is as defined in Eq. 7.6.

Subsequently, we select the WS to query which has the maximum $\mathcal{VOC}$, analogously to Section 4.4. However, the query is issued only if the $\mathcal{VOC}^*$ is greater than the utility of the query cost to the designer. Formally if $\mathcal{VOC}^*(s) > U(QueryCost)$, the the query is issued to the provider whose service led to the maximum expected change.

Potentially, the value of $VOC^*(s)$ and $\mathcal{VOC}^*(s)$ may be different and the service targeted for querying, a^* will change.

In our example, the composition of a risk-averse manufacturer may be more sensitive to changes in the parameters of the Spot Market. This is because the risk-averse manufacturer

could be relying on the Spot Market to satisfy its parts order. This is in contrast to a risk-neutral manufacturer whose composition is expected to be most affected by changes in the parameters of the Preferred Supplier.

7.4 Performance Evaluation

We evaluate the performance of our risk-sensitive VOC and subsequent adaptation of the compositions in the context of the risk-sensitive supply chain of Section 7.1. Specifically, compositions adapted using VOC lead to significantly better performance in volatile environments compared to compositions that are not adapted and to those that are adapted using adhoc techniques. Hence, I will not demonstrate the benefits of VOC here; rather I will focus on illustrating the influence of risk preferences on composition and adaptation, and thereby demonstrate the intuitive validity of the general approach.

As mentioned previously, in the absence of risk preferences a rational manufacturer would choose the preferred suppliers Web service to order parts. Given the volatility in the environment (see μ and σ values in Figure 7.1), let the manufacturer use a VOC-driven approach toward querying service providers for revised information. If the updated information leads to a change in the optimal policy, the composition is adapted. In this context, we show the performance of the risk-neutral manufacturers compositions in Figure 7.3. The average reward obtained by three distinct adaptive compositions that invoke the inventory, preferred supplier and spot market WSs, respectively is shown. Here, each data point is the average of 500 executions of the composition in 1,000 simulations of our problem domain. The simulations are constructed by sampling the Gaussian distributions of parameters of the participating volatile WSs. Each of the three different compositions experienced identical simulations in order to facilitate a valid comparison. A possible VOC-driven query is issued at the starting state of the composition.

The first notable observation is that all three compositions show a drop in expected reward as the query cost increases. This is typical of VOC-driven adaptations. While adap-

tations are frequent when the query cost is low, they occur less as querying becomes more expensive. Notice that the composition prescribing the invocation of the preferred supplier continues to do the best. This implies that for low query costs, possible adaptations continue to outperform those in compositions that prescribed invoking other WSs. Furthermore, the risk-neutral manufacturer queried the preferred supplier the most because the associated VOC was often the largest. For large query costs, there is no adaptation and it is rational to choose the preferred supplier.

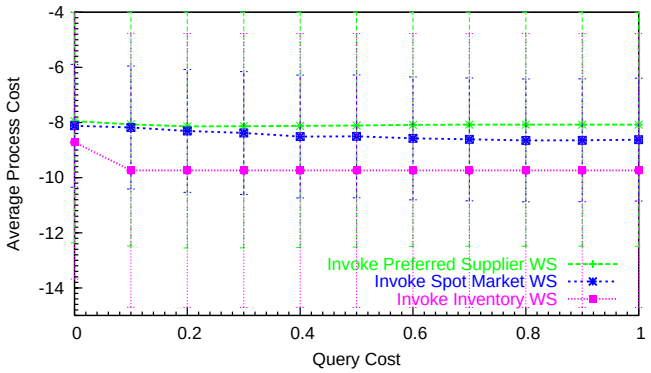


Figure 7.3: VOC based adaptive compositions for a risk-neutral manufacturer. We compare a manufacturer for policies that select Preferred Supplier (shown in green), Spot Market (blue), and Inventory (purple) services. Lower average process cost indicates better performance. The deviation bars demonstrate the variance caused by a randomized environment.

Figure 7.4, demonstrates how the adaptive compositions of a risk-seeking manufacturer would perform when using VOC-driven selective querying. The methodology for generating the data is same as before except for γ which is 1.5 and we show the utility of the reward obtained by the compositions. Risk-sensitive VOC was computed according to Eq. 7.7. As mentioned previously, a risk-seeking manufacturer opts to invoke the inventory in comparison to a rational manufacturer who chooses the preferred supplier. Figure 7.4 demonstrates that possible adaptations of this composition have a larger utility to the risk-seeking manufacturer than the adaptive preferred supplier based WS composition, which is the choice of a rational manufacturer. Thus, although the composition that prescribes the inventory performs worst in the absence of risk preferences, it becomes the optimal choice for a risk seeker.

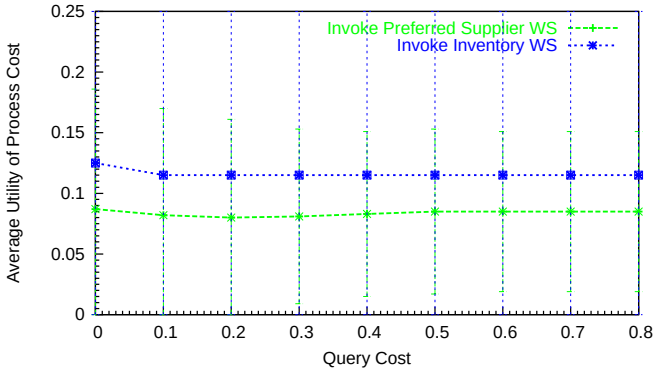


Figure 7.4: VOC based adaptive compositions for a risk-seeking manufacturer ($\gamma = 1.5$). We compare a manufacturer for policies that select Preferred Supplier (shown in green) and Inventory (blue) WSs. Higher average utility indicates better performance. The deviation bars demonstrate the variance caused by a randomized environment.

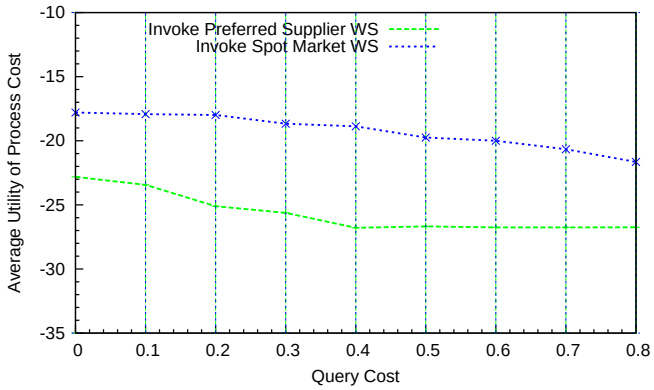


Figure 7.5: VOC based adaptive compositions for a risk-averse manufacturer ($\gamma = 0.75$). We compare a manufacturer for policies that select Preferred Supplier (shown in green) and Spot Market (blue) WSs. Higher average utility indicates better performance. The deviation bars demonstrate the variance caused by a randomized environment.

Observe that for low query costs, the risk-seeking manufacturer queried the inventory the largest number of times indicating that its VOC was the largest. This is because of the high standard deviation of its cost coupled with its importance in the composition of the manufacturer. About a fraction of one-tenth of these queries led to adaptations where the

preferred supplier was selected. Finally, in Fig. 5 we demonstrate the utility to a risk-averse manufacturer of the performance of adaptive compositions. We use = 0.75 (recall that risk aversion is modelled using < 1) and each data point is generated as before. In comparison to a risk seeker, our risk-averse manufacturer continues to find the adaptive composition that recommends invoking the spot market most preferable. Its utility remains consistently high compared to that of the adaptive composition that involves the preferred supplier which is the choice of a rational manufacturer. Furthermore, the risk-averse manufacturer queried the preferred supplier most number of times. This is intuitive because while the spot market is important its deviation is very low. Significant changes in preferred suppliers costs could make it the WS of choice for the risk-averse manufacturer. Despite the querying the number of times that adaptations do occur is low because of the significant value of the spot market to the risk-averse manufacturer. About one-fifth of these queries led to adaptations. In summary, risk-sensitive VOC leads to different services being selected for querying based on the risk preferences, in comparison to queries in the absence of risk. This is in part due to the varying compositions induced by different risk preferences.

7.5 Summary

Although designers are often advised to be rational and objective while designing processes, risk preferences invariably play a role in the composition. Therefore, I presented an approach for considering risk preferences while composing and querying, leading to the risk sensitive VOC. I used utility functions to model risk preferences and showed how these may be integrated with the traditional VOC. Our experimental results on a simulated parts procurement domain confirmed the intuition about the impact of risk its consideration leads to changes in how we compose and issue queries compared to rational behavior. While I experimented with a simple scenario in order to promote clarity, I think that our results are indicative of more complex scenarios as well. Although I used exponential utility functions, other forms of utility functions could be used as well such as those which allow a switch in risk preference

depending on the accumulated reward [42]. Finally, it would be interesting to demonstrate the beneficial role of risk in real-world compositions. In this regard, a more complex case study should be sought.

Chapter 8

Discussion and Future Work

WSC environments are often volatile – parameters of component services may vary over time. In order to remain cost-effective in such environments, WSCs should adapt by maintaining an up-to-date account of the revised parameters. Obtaining the revised parameter information requires querying the component services for their revised parameters so that the composition model contains up-to-date knowledge of those parameters. While querying provides us with the information we need to construct a more optimal process, we found it was optimal to design a querying strategy that considered several factors in order to be effective. Specifically, an optimal querying strategy should: (1) determine if the revised parameter information of a service is *useful* and *cost-efficient*, (2) be *computationally tractable*, (3) understand how to query in order to adapt *nested* configurations of services in a WSC, and (4) allow for considerations of a composition user's *risk preferences*. This thesis addressed those concerns.

Chapter 4 introduced a method called the value of changed information (VOC) method, that queries and adapts a WSC to changes in parameters of the services, only if the revised parameters are worth obtaining. VOC specifically enables us to know: (1) when is it cost effective to query for the changed information and, (2) which service(s) to query. VOC can seamlessly be integrated into any SOA, and can be added to composition algorithms with little effort. As a result, the WSC can adapt more intelligently and at lesser cost than querying with naive and generic heuristics. This was made evident by empirical evaluations in which the VOC-driven querying strategy outperformed other commonly used naive querying strategies in terms of lowering the overall average cost of executing a WSC.

As VOC is an expensive computation, Chapter 5 outlined ways to mitigate the complexity of computing VOC so that this method of adaptation becomes feasible. We identified those parameter values that are not expected to cause changes in the composition and may be ignored while computing the VOC. We then were able to exploit service parameter guarantees during which parameters' values remain unchanged. Services whose parameter guarantees have not expired need not be considered for querying. We demonstrated the speedups when using these techniques both empirically and through theoretical complexity analysis.

Chapter 6 demonstrated that the VOC-driven querying stragegy can be applied to WSCs that contain services assembled in hierarchical WSCs. This is significant because large, real-world WSCs often tend to have such complex configurations. Two of the primary challenges that we addressed are how to obtain beliefs over volatility of composite WS parameters, and which of the component WSs to invoke if a composite WS is found to potentially cause the most expected change in the composition. As a result, we maintained the advantages of using VOC over other querying methods in more complex compositions.

While previously, VOC was applied toward adapting WSCs indifferent to risk, Chapter 7 extended its applicability to adapt to risk-sensitive WSCs. We found that as we vary the degree of risk sensitivity, we affected which WSs would be included for participation in compositions, and which WSs are selected for querying of revised information.

8.1 Future Work

While this thesis has overcome the major challenges of adapting WSCs through the use of intelligently querying for revised information, there are many open avenues for future improvement. I outline some of those here.

Case studies Although I have outlined several scenarios across many different domains (Chapter 3), stronger case studies would provide further evidence of the validity of VOC. This is especially true for the implementation of risk-sensitive VOC, whose improved performance would be more readily apparent on processes that are implemented on a larger scale.

Implementation of VOC as an enhancement to HALEY A VOC-based querying mechanism can easily enhance a readily available composition tool, such as HALEY [85], a hierarchical framework for logically composing Web services created here at the University of Georgia. This is currently an open project.

Non-myopic approaches for information revision Non-myopic querying methods would improve the accuracy with which VOC measues the value of change. As non-myopic approaches have been proposed, they add a higher degree of complexity. This complexity must therefore be accounted for so that it may be utilized a viable solution for adaptation.

Adapting compositions with volatile qualititative properties Currently, VOC can only be applied to *quantitative* WS parameters such as cost and availability. Equally important are *qualitative* parameters such as WS trust, which impact on the effectiveness of a composition as well. As this line of research matures, we should find ways of incorporating qualitative parameters such as trust in computing VOC.

Using VOC in RESTful Web service compositions The approaches in this thesis have assumed SOAs that function in SOA environments consisting of traditional, operations-based WSDL services. However, RESTful [29], or resource-centric WSs, are quickly becoming a popular architectural style for WS implementation. While research in RESTful composition methods (e.g. [86]) are in its infancy, it is important to consider the problem of volatility in these compositions as well.

These improvements, and likely more, should be taken into consideration so as to build a truly optimal and efficient querying-based system suitable for adapting all types of Web service compositions. Nevertheless, this thesis takes the necessary first step in realizing such a system.

Bibliography

[1] Active Endpoints. ActiveBPEL Engine [online]. 2007. Available from: http://www.active-endpoints.com/ [cited June 10, 2010].

[2] R. Aggarwal and K. Verma and J. Miller and W. Milnor. Constraint driven Web service composition in METEOR-S. *IEEE International Conference on Service Computing*, pages 23–30, 2004.

[3] V. Agarwal, G. Chafle, K. Dasgupta, N. Karnik, A. Kumar, S. Mittal, and B. Srivastava. Synthy: A System for End to End Composition of Web Services. *Journal of Web Semantics*, Vol 3, No 4, pages 311–339, 2005.

[4] V. Agarwal, G. Chafle, S. Mittal, and B. Srivastava. Understanding Approaches for Web Service Composition and Execution *Bangalore Annual Compute Conference (COMPUTE 2008)*, pages 1–8, 2008.

[5] Amazon Web Services Available from: http://aws.amazon.com/ [June 10, 2010].

[6] Apache Software Foundation. Axis2 [online]. Available from: http://ws.apache.org/axis2/ [June 10, 2010].

[7] Apache Software Foundation. Tomcat [online]. Available from: http://tomcat.apache.org/ [June 10, 2010].

[8] T. Au, D. Nau, and V.S. Subrahmanian. Utilizing Volatile External Information During Planning. *Eureopean Conference on Artificial Intelligence (ECAI)*, pages 647–651, 2004.

[9] T. Au, U. Kuter, and D. Nau. Web Service Composition with Volatile Information. *International Semantic Web Conference (ISWC)*, pages 52–66, 2005.

[10] T. Au and D. Nau. Reactive Query Policies: A Formalism for Planning with Volatile External Information. *IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*, pages 243–250, 2007,

[11] M. Avila-Godoy. Controlled Markov Chains with Exponential Risk-Sensitive Criteria: Modularity, Structured Policies and Applications. Ph.D. thesis, Department of Mathematics, University of Arizona, 1999.

[12] B. Bojanov, H. Halcopian and A. Sabakian, Spline Functions and Multivariate Interpolations. *Kluwer Academic Publishers*, 1993.

[13] Tolerating Exceptions in Workflows: a Unified Framework for Data and Processes. *International Joint Conference on Work Activities Coordination and Collaboration*, pages 59–68, 1999.

[14] Web Services Business Process Execution Language Version 1.1 Specification. Available from: http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel/ws-bpel.pdf [June 10, 2010].

[15] Web Services Business Process Execution Language Version 2.0 Specification. Available from: http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf [June 10, 2010].

[16] M. Brambilla, S. Ceri, S. Comai, and C. Tziviskou. Exception Handling in Workflow-Driven Web Applications. *World Wide Web Conference (WWW)*, pages 170–179, 2005.

[17] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. *WS-Agreement Specification*, 2005.

[18] J. Cardoso, A. Sheth, J. Miller, and J. Arnold. Quality of Service for Workflows and Web Service Processes. *Journal of Web Semantics*, Vol 1, No 3, pages 281–308, 2004.

[19] F. Casati and M.Shan. Event-Based Interaction Management for Composite E-Services in eFlow. *Information Systems Frontiers*, Vol. 4, No. 1, pages 19–31, 2002.

[20] G. Chafle, K. Dasgupta, A. Kumar, S. Mittal, and B. Srivastava. Adaptation in Web Service Composition and Execution. In *International Conference on Web Services (ICWS), Industry Track*, pages 549–557, 2006.

[21] G. Chafle, P. Doshi, J. Harney, S. Mittal, and B. Srivastava. Improved Adaptation of Web Service Compositions Using Value of Changed Information. *International Conference on Web Services (ICWS), Industry Track*, pages 784–791, 2007.

[22] A. Charfi and M. Mezini. Aspect-Oriented Web Service Composition with AO4BPEL. *European Conference on Web Services*, pages 168–182, 2004.

[23] J. Corner and P. Corner. Characteristics of Decisions in Decision Analysis Practice. In *The Journal of Operational Research Society*, Vol. 46, pages 304–314, 2006.

[24] A. Mallya, N. Desai, A. Chopra, and M. Singh. OWL-P: A Methodology for Business Process Development. *Agent Oriented Information Systems*, pages 79–94, 2005.

[25] N. Desai, A. Chopra and M.P. Singh. Business Process Adaptations via Protocols. *International Conference on Services Computing*, pages 601–608, 2006.

[26] K. Dickson, W. Chiu, Q. Li, and K. Karlapalem. A Meta Modeling Approach to Workflow Management Systems Supporting Exception Handling. *Information Systems*, Vol. 24, No. 2, pages 159–184, 199.

[27] P. Doshi, R. Goodwin, R. Akkiraju, and K. Verma. Dynamic Workflow Composition Using Markov Decision Processes. *Journal of Web Services Research*, Vol. 2, No. 1, pages 1–17, 2005.

[28] C. Ellis, K. Keddara, and G. Rozenberg. Dynamic Change Within Workflow Systems. *ACM Conference on Organizational Computing Systems*, pages 10–21, 1995.

[29] R. Fielding. Architectural Styles and the Design of Network-based Software Architecture. *PhD thesis*, University of California, Irvine, 2000.

[30] A. Glen, L. Leemis, and J. Drew. Computing the Distribution of the Product of Two Continuous Random Variables. *Computational Statistics and Data Analysis (CSDA)*, Vol. 44, No. 3, pages 451–464, 2004.

[31] K. Gomadam, A. Ranabahu, L. Ramaswamy, A. Sheth, and K. Verma. A Semantic Framework for Identifying Events in a Service Oriented Architecture. *International Conference on Web Services*, pages 545–552, 2007.

[32] D. Gotz and K. Mayer-Patel. A General Framework for Multidimensional Adaptation. *IEEE International Conference on Multimedia and Expo (ICME 2004)*, pages 612–619, 2004.

[33] J. Harney and P. Doshi. Adaptive Web Processes Using the Value of Changed Information. *International Conference on Service-Oriented Computing (ICSOC)*, pages 179–190, 2006.

[34] J. Harney and P. Doshi. Speeding Up Adaptation of Web Service Compositions Using Expiration Times. *World Wide Web (WWW)*, pages 1023–1032, 2007.

[35] J. Harney and R. Doshi. Speeding Up Web Service Composition with Volatile External Information. *International Workshop on Context enabled Source and Service Selection, Integration, and Adaptation (CSSSIA)*, pages 1201–1202, 2008.

[36] J. Harney and P. Doshi. Selective Querying for Adapting Web Service Compositions Using the Value of Changed Information. *IEEE Transactions on Services Computing, in press*, 2008.

[37] He, Qiang and Yan, Jun and Jin, Hai and Yang, Yun. Adaptation of Web Service Composition Based on Workflow Patterns. *International Conference on Service-Oriented Computing (ICSOC)*, pages 22–37, 2008.

[38] S. Holtzman. Intelligent Decision Systems. *Addison-Wesley*, 1989.

[39] C. Kirkwood. Approximating Risk Aversion in Decision Analysis Applications. *Decision Analysis*, pages 55–72, 2004.

[40] N. Kokash and V. D'Andrea. Evaluating Quality of Web Services: A Risk-Driven Approach. *Business Information Systems*, 2007, pages 180–194.

[41] Y. Liu. Decision-Theoretic Planning Under Risk-Sensitive Planning Objectives. Ph.D. thesis, College of Computing, Georgia Institute of Technology, 2005.

[42] Y. Liu. Risk-Sensitive Planning with One-Switch Utility Functions: Value Iteration *AAAI Conference on Artificial Intelligence*, 2005, 993–999.

[43] Z. Luo, A. Sheth, K. Kochut, and J. Miller. Exception Handling in Workflow Systems. *Journal of Applied Intelligence*, Vol 13, No 2, 125–147, 2000.

[44] Z. Maamar, N. Narendra, D. Benslimane, and S. Sattanathan. Policies for Context-Driven Transactional Web Services. *International Conference on Advanced Information Systems Engineering (CAiSE 2007)*, pages 249–263, 2007.

[45] T. Mitchell Machine Learning. *McGraw-Hill*, 1997.

[46] M. zur Muehlen and D. Ho. Risk Management in the BPM Lifecycle. *Business Process Management Workshops*, pages 454–466, 2005.

[47] R. Muller, U. Greiner, and E. Rahm. Agentwork: a Workflow System Supporting Rule-based Workflow Adaptation. *Journal of Data and Knowledge Engineering*, Vol 51, No 2, 223–256.

[48] N. Narendra and S. Gundugola. Automated Context-Aware Adaptation of Web Service Executions. *ACS/IEEE International Conference on Computer Systems and applications (AICCSA 2006)* ,pages 179–187, 2006.

[49] N. Narendra, K. Ponnalagu, J. Krishnamurthy, and R. Ramkumar. Run-Time Adaptation of Non-functional Properties of Composite Web Services Using Aspect-Oriented

Programming. *International Conference on Services Oriented Computing*, pages 546–557, 2007.

[50] T.C. Au, U. Kuter, and D. Nau. Web Service Composition with Volatile Information. In *International Semantic Web Conference (ISWC)*, 2005, pages 52–66.

[51] T.C. Au and D. Nau. Reactive Query Policies: A Formalism for Planning with Volatile External Information. *IEEE Symposium on Computational Intelligence and Data Mining*, pages 243–250, 2007.

[52] H. Paques, L. Liu, and C. Pu. Adaptation Space: A Design Framework for Adaptive Web Services. *International Journal of Web Services Research*, Vol 1, No 3, 2004.

[53] S. Paradesi, P. Doshi, and S. Swaika. Integrating Behavioral Trust in Web Service Compositions. *International Conference on Web Services (ICWS 2009)*, pages 453–460, 2009.

[54] J. Pathak, S. Basu, R. Lutz and V. Honavar. An Approach for Composing Web Services Through Iterative Reformulation of Functional Specifications. In *International Journal on Artificial Intelligence Tools)*, Vol. 17, No. 1, pages 109–138, 2008.

[55] M. Pistore, P. Traverso, P. Bertoli, and A. Marconi. Automated Synthesis of Composite BPEL4WS Web Services *IEEE International Conference on Web Services (ICWS 2005)*, pages 293–301, 2005.

[56] M. L. Puterman. *Markov Decision Processes:Discrete Stochastic Dynamic Programming.* Wiley series in probability and mathematical statistics. Wiley-Interscience, 1994.

[57] M. Reichert and P. Dadam. Adeptflex-supporting Dynamic Changes of Workflows without Losing Control. *Journal of Intelligent Information Systems*, Vol 10, No 2, pages 93–117, 1998.

[58] V.K. Rohatghi. An Introduction to Probability Theory and Mathematical Statistics. *John Wiley & Sons*, 1976.

[59] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach (Second Edition)*. Prentice Hall, 2003.

[60] The Supply-Chain Council. Available from: http://archive.supply-chain.org/cs/root/scor_tools_resources/scor_model/scor_model [cited June 10, 2010).

[61] A. Sheth, Y. Han, and C. Bussler. A Taxonomy of Adaptive Workflow Management *CSCW-98 Workshop, Towards Adaptive Workflow Systems*, 1998.

[62] Qos for Service-oriented Middleware. *Multiconference on Systemics Cybernetics and Informatics* Vol 8, pages 528–534, 2002.

[63] E. Sirin, B. Parsia, D. Wu, J. A. Hendler, and D. S. Nau. Htn Planning for Web Service Composition using Shop2. *Journal of Web Semantics*, 1(4):377–396, 2004.

[64] U. Kuter, E. Sirin, D. S. Nau, B. Parsia, and J. A. Hendler. Information Gathering During Planning for Web Serivce Composition. *Journal of Web Semantics*, 2(2–3):183–205, 2005.

[65] E. Stohr and J.Zhao. A Technology Adaptation Model for Business Process Automation. *Hawaii International Conference on System Sciences*, pages 405–414, 1997.

[66] D. Strong and S. Miller. Exceptions and Exception Handling in Computerized Information Processes. *ACM Trans. Inf. Syst.*, Vol 13, No 2, pages 206–233, 1995.

[67] W.M.P van der Aalst. Exterminating the Dynamic Change Bug: A Concrete Approach to Support Workflow Change. *Information Systems Frontiers*, Vol 3, No 3, pages 297–317, 2001.

[68] W.M.P van der Aalst and S. Jablonski. Dealing with Workflow change: Identification of Issues and Solutions *International Journal of Computer Systems Science and Engineering*, Vol 5, pages 267–276, 2000.

[69] K. Verma, P. Doshi, K. Gomadam, J. Miller, and A. Sheth. Optimal Adaptation in Web Processes with Coordination Constraints. *International Conference on Web Services*, pages 257–264, 2006.

[70] J. von Neumann, and O. Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1944.

[71] vRESCo - Vienna Runtime Environment for Service-oriented Computing Available from: http://berlin.vitalab.tuwien.ac.at/prototypes/vresco/ [cited June 10, 2010].

[72] W. Wiesemann, R. Hochreiter, and D. Kuhn. A Stochastic Programming Approach for QoS-Aware Service Composition. In *IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2008)*, pages 226–233, 2008.

[73] W3C. SOAP Version 1.2, 2003. Available from: http://www.w3.org/TR/soap [cited June 10, 2010].

[74] UDDI Version 3.0.2, 2004. Available from: www.uddi.org/pubs/uddi_v3.htm [cited June 10, 2010].

[75] W3C. Web Services Description Language (WSDL) 1.1, 2001. Available from: http://www.w3.org/TR/wsdl [cited June 10, 2010].

[76] W3C. Web Services Description Language (WSDL) 2.0, 2007. Available from: http://www.w3.org/TR/wsdl20/ [cited June 10, 2010].

[77] W3C. Web Services Policy Framework v1.5, Sept. 2007. Available from: http://www.w3.org/TR/ws-policy/ [cited June 10, 2010].

[78] D. Wu, B. Parsia, E. Sirin, J. Hendler, and D. Nau. Automating DAML-S Web Services Composition Using SHOP2. *International Semantic Web Conference*, pages 195–210, 2003.

[79] Y. Wu and P. Doshi. Regret-Based Decentralized Adaptation of Web Processes with Coordination Constraints. *International Conference on Services Computing*, pages 262–269, 2007.

[80] Microsoft XBOX Supply Chain. Enabling an Adaptable, Aligned, and Agile Supply Chain with BizTalk Server and RosettaNet Accelerator. Available from: *http:// www.microsoft.com / technet / itshowcase /content / scmbiztalktcs.mspx*, 2005. [cited June 10, 2010].

[81] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Quan Sheng. Quality Driven Web Services Composition. *World Wide Web*, pages 411–421, 2003.

[82] L. Zeng, H. Lei, and H. Chang. Monitoring the QoS for Web Services. *International Conference on Service-Oriented Computing (ICSOC07)*, pages 132–144, 2007.

[83] Y. Zhai., J. Zhang, and K. Lin. SOA Middleware Support for Service Process Reconfiguration with End-to-End QoS Constraints *IEEE International Conference on Web Services (ICWS 2009)*, pages 815–822. 2009.

[84] H. Zhao and P. Doshi. A Hierarchical Framework for Composing Nested Web Processes. *International Conference on Services Oriented Computing*, pages 116–128, 2006.

[85] H. Zhao and P. Doshi. Haley: An End-to-end Scalable Web Service Composition Tool. In *Developer's Track, World Wide Web (WWW)*, 2008.

[86] H. Zhao and P. Doshi. Towards Automated RESTful Web Service Composition/ In *Research Track, International Conference on Web Services (ICWS)*, pages 189–196, 2009.

Appendix

Sample BPEL Code for Adaptive MSXBOX Supply Chain

This section examines the generated BPEL code for the MSXBOX Supply Chain in more detail [1]. Specifically, some of the important concepts demonstrated in Figures 4.3 and 4.4 and in Section 4.3 will be highlighted. The BPEL as described by this section was used in specifically implementing the MSXBOX Supply Chain example, however, it may be analogously applied to other scenarios as well. It is important to note that this architecture serves as only a base implementation, as other components and configuration files may be required in addition to the architecture described in Section 4.3, depending on the requirements of the composition that is being designed. This is due to the many variations of composition techniques, BPEL engines and implementation tools that are available. Nevertheless, the procedures required to utilize VOC in compositions remains relatively consistent despite any system configuration.

An overview of the BPEL document can be seen in Figure 1. For ease of illustration, I leave out descriptions for the process definitions, import statements, and partner links. The information in these sections should be created in accordance to the architecture described in Figure 4.4.

The variable definitions for the composition are shown in Figure 2. Each of the variables shown here are essential for composition with VOC[2]. Included are variables representing the state (i.e. $GPUInventorySupplierStatus$, $GPUPreferredSupplierStatus$, and

[1]Note that the full BPEL document as well as other information regarding VOC will soon be made available through the THINC Lab web site (URL: http://thinc.cs.uga.edu).

[2]Note that other variables could be used as well, such as temporary local variables for the composition.

BPEL Overview

```
<!-- Process Definitions -->
 <bpel:process … />
<!-- Import Statments -->
      <bpel:import>
      …
       </bpel:import>
<!-- PartnerLinks -->
      <bpel:partnerLinks>
      …
       </bpel:partnerLinks>
<!-- Variables -->
       <bpel:variables>
       …
        </bpel:variables>

 <!-- Process Logic -->
        <bpel:sequence>
                <!-- MSXBOX Supply Chain -->
        </bpel:sequence>
</bpel:process>
```

Figure 1: An overview of the BPEL document.

$GPUSpotMarketSupplierStatus$), optimal policy ($Policy$), a boolean representing whether or not the goal has been reached ($Goal$), and a variable maintaining the cost of querying ($QueryCost$). $Policy$ is an array whose size is the number of possible states in the composition. Each element in the array corresponds to an optimal action to perform at a specific state. In addition, each of the variables are initialized (not shown in the figure). Typically, $Goal$ is set to $false$, the state variables are set to $Unknown$, and the $Policy$ is taken from the input given to the BPEL process.

BPEL Overview for the Variables in the MSXBOX Supply Chain

```
<!-- Process Definitions -->
<!-- Import Statments -->
<!-- PartnerLinks -->
<!-- Variables -->
    <bpel:variables>

        <bpel:variable name="GPUInventorySupplierStatus" type="xsd:string"/>
        <bpel:variable name="GPUPreferredSupplierStatus" type="xsd:string"/>
        <bpel:variable name="GPUSpotMarketSupplierStatus" type="xsd:string"/>

        ...

        <bpel:variable name="Goal" type="xsd:boolean"/>
        <bpel:variable name="Policy" messageType="XBOX:PolicyMessage"/>
        <bpel:variable name="QueryCost" type="xsd:int"/>

    </bpel:variables>
<!-- Process Logic -->
```

Figure 2: The BPEL variables in the MSXBOX supply chain.

Figure 3 demonstrates the general logic needed to implement the composition algorithm demonstrated in Section 4.2 (see Figure 4.2). The composition is wrapped in a $< bpel : while >$ activity, which terminates when the *Goal* variable is set to *true*. Within the $< bpel : while >$ activity, the composition determines which state the composition currently resides. The $< bpel : if >$ and $< bpel : condition >$ statements branch the process according to the current state. Note that state is represented by a conjunction of variable assignments to the state variables, $GPUInventorySupplierStatus$, $GPUPreferredSupplierStatus$, and $GPUSpotMarketSupplierStatus$. Demonstrated in the figure is one such state where $GPUInventorySupplierStatus$, $GPUPreferredSupplierStatus$, and $GPUSpotMarketSupplierStatus$ are all $Unknown$. If the composition is in this state, we proceed by invoking the VOC WS and subsequently the external WS (determined by

Policy) for that state. These steps can be followed analgously for each of the states in the composition.

Figure 4 demonstrates how the policy can be used to invoke external WSs. Once the state has been determined, the *Policy* variable is used to find the optimal WS to be invoked for that specific state. The WS contained in the state's index of the array is chosen for invocation. The $< bpel : invoke >$ activity is utilized on the partner service's *invoke* operation [3], where the output is incorporated into the state variables to form a new state. For example, invoking **CheckGPUInventorySupplierStatus** may change the variable $GPUInventorySupplierStatus$ from $Unknown$ to Yes, effectively transitioning to a new state in the composition. The composition will then branch to that state in the next iteration of the $< bpel : while >$ activity. Again, if this particular state is a goal state, the *Goal* variable will be set to true.

Finally, Figure 5 shows the BPEL needed to utilize VOC. When BPEL branches into a state, the VOC procedure is used before an invocation of an external WS provider. First, the VOC service is invoked, where the *vocOutput* variable carries information about the value for VOC^* and the corresponding WS, a^* (see Section 4.1). VOC^* is compared to the global $QueryCost$ variable. If $VOC^* > QueryCost$, then we progress to invoking the revised information providing service operation, $getRevisedData$. This service operation is assumed to be placed in the same $portType$ as the *invoke* service described above. Once the data is retrived, it is used as input to the invocation of the $PolicyGenerator$ WS, which computes the new policy required for the composition. If $VOC^* \leq QueryCost$ the process bypasses the query service and invoke the WS currently prescribed by *Policy*. Then, the state is reformulated and the composition continues.

[3]The specific name *invoke* may differ from composition to composition. For illustration purposes, it is used as the name here.

BPEL Overview for the Composition Algorithm

```
<!-- Process Definitions -->
<!-- Import Statments -->
<!-- PartnerLinks -->
<!-- Variables -->
<!-- Process Logic -->
  <bpel:sequence>
    <bpel:receive createInstance="yes" name="start"
                operation="startSupplyChain"
                partnerLink="MSXBOXSupplyChainPartnerLink"
                portType=XBOX:MSXBOXSupplyChainPortType"
                variable="Policy"/>

      <!-- Continue composition until the goal has been reached -->
      <bpel:while>
          <bpel:condition>
              $Goal = 'false'
          </bpel:condition>

           <!-- if in state 1... -->
          <bpel:if>
            <bpel:condition>
                    $GPUInventorySupplierStatus = 'Unknown' and
                    $GPUPreferredSupplierStatus = 'Unknown' and
                    $GPUSpotMarketSupplierStatus = 'Unknown'
             <bpel:condition>

            <!-- Run voc algorithm and invoke service here  -->


             <!-- if state 2... -->
             <bpel:elseif>
                    …
             </bpel:elseif>

            <!-- if state n… -->
            <else>
                    ...
            </else>
      </bpel:while>

    <bpel:reply name="end"
                operation="startSupplyChain"
                partnerLink="MSXBOXSupplyChainPartnerLink"
                portType="test:MSXBOXSupplyChainPortType"
                variable="response"/>

  </bpel:sequence>
```

Figure 3: An overview of the composition algorithm.

BPEL Overview for Invoking Web services

```
<!-- Process Definitions -->
<!-- Import Statments -->
<!-- PartnerLinks -->
<!-- Variables -->
<!-- Process Logic -->
   <bpel:sequence>
     <bpel:receive … />
        <bpel:while>
            <bpel:condition  …. />
             <!-- if in state 1... -->
            <bpel:if>
              <bpel:condition>  $GPUInventorySupplierStatus = 'Unknown' and
                                $GPUPreferredSupplierStatus = 'Unknown' and
                                $GPUSpotMarketSupplierStatus = 'Unknown'
             <bpel:condition>

              <!-- Use VOC to query for revised data if needed -->

              <!-- Use policy to invoke a Web service -->

              <bpel:sequence>
                    <bpel:if>
                          $Policy/PolicyPart/ws[1] = 'CheckGPUInventorySupplierStatus'
                    </bpel:if>

                    <!-- invoke the service partner -->
                    <invoke name="CheckGPUInventorySupplierStatus"
                        partnerLink="CheckGPUInventorySupplierStatusPartnerLink"
                        portType="inv:CheckGPUInventorySupplierStatusPortType"
                        operation="invokeGPUInv" inputVariable="GPUInventoryInput"
                        outputVariable="GPUInventoryOutput" />

                    <!-- Reformulate the state -->
                   <bpel:assign>
                      <bpel:from variable="GPUInventoryOutput" />
                      <bpel:to variable="GPUInventorySupplierStatus" />
                   <bpel:assign>

                    <bpel:elseif>
                          <!-- check if the polcy recommends a difference servie -->
                    </bpe:elseif>
                    …
                    <bpel:else>
                    </bpel:else>

                    <!-- if the goal state has been reached at this point, change $Goal to 'true' -->
              </bpel:sequence>

            </bpel:sequence>
           <!-- other states -->
           <bpel:elseif … />
           <bpel:else … />
        </bpel:while>
     <bpel:reply ..."/>
   </bpel:sequence>
```

Figure 4: An overview of how BPEL uses the policy to invoke Web services offered by external service partners.

BPEL Overview for Invoking the VOC Service

```
<!-- Process Definitions -->
<!-- Import Statments -->
<!-- PartnerLinks -->
<!-- Variables -->
<!-- Process Logic -->
   <bpel:sequence>
      <bpel:receive … />
         <bpel:while>
            <bpel:condition  …. />
             <!-- if in state 1... -->
            <bpel:if>
              <bpel:condition>  $GPUInventorySupplierStatus = 'Unknown' and
                                        $GPUPreferredSupplierStatus = 'Unknown' and
                                        $GPUSpotMarketSupplierStatus = 'Unknown'
             <bpel:condition>
             <bpel:sequence>
                <!-- Run voc algorithm and invoke service here  -->
                <!-- Invoke the VOC service -->
                <invoke name="VOCServiceState1"
                 partnerLink="VOCPartnerLink" portType="voc:VOCPortType"
                 operation="getVOC" inputVariable="vocInput" outputVariable="vocOutput">

              <bpel:if>
                  <bpel:condition> $vocOutput &gt; $QueryCost </bpel:condition>
                  <bpel:sequence>

                     <!-- invoke the revised data extraction operation from the VOC* service -->
                     <invoke name="GPUInventoryQueryServiceState1"
                              partnerLink="GPUPartnerLink" portType="inv:GPUInventoryPortType"
                              operation="getRevisedData" inputVariable="GPUInvInput"
                               outputVariable="GPUInvOutput">

                     <!-- input GPUInvOutput into the policy generator service-->
                     <invoke name="PolicyGenState1" partnerLink="PolicyGenPartnerLink"
                                    portType="pg:PolicyGenPortType" operation="resolvePolicy"
                              inputVariable="pgInput" outputVariable="pgOutput">

                     <!-- Assign the output of the Policy Generator to the Policy variable -->
                     <bpel:assign>
                        <bpel:copy>
                              <bpel:from variable="pgOutput"/>
                              <bpel:to variable="Policy"/>
                        </bpel:copy>
                     </bpel:assign>
                  </bpel:sequence>
               <!-- else no query is performed -->

               <!-- Use policy to invoke a Web service -->

             </bpel:sequence>
            <!-- other states -->
            <bpel:elseif … />
            <bpel:else … />
         </bpel:while>
      <bpel:reply ..."/>
   </bpel:sequence>
```

Figure 5: An overview of the procedure of invoking the VOC service.