FAKULTÄT FÜR !NFORMATIK

Quality of Service Driven Workflows within the Microsoft .NET Environment

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering/Internet Computing

eingereicht von

Alexander Schindler, Bakk.techn.
Matrikelnummer 9926045

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung:
Betreuer: Univ.Prof. Dr. Schahram Dustdar
Mitwirkung: Univ.Ass. Dr. Florian Rosenberg

Wien, 19.10.2009

(Unterschrift Verfasser) (Unterschrift Betreuer)

Abstract

Service-oriented architecture (SOA) is gaining more and more momentum in todays software engineering, for its ease of integrating heterogeneous systems. Web services allow for building complex and dynamic systems where single components can be discovered at run-time by means of certain criteria like Quality of Service (QoS) information. QoS-aware service management and composition highly depends on this valuable data, but most of the currently proposed solutions assume that this information is readily available.

This master thesis addresses this fundamental issue by providing two methods to monitor the performance of WCF Web services and composite Web services - also called workflows. One is based on Windows Performance Counters (WPC) provided by the Windows Communication Foundation (WCF). These counters provide a highly accurate way to measure QoS data. The retrieved information is aggregated and attached as valuable metadata to the revision datasets which are stored in the VRESCo registry. The second method monitors Windows Workflow Foundation (WWF) workflows. By taking also complementary activities of workflow into account, a more difficile view of its actual performance is being retrieved. Such information is invaluable for QoS aggregation algorithms. The thesis also introduces the current state of the art of service-oriented architecture and related technology and gives an overview of relevant related work in this field.

Zusammenfassung

Das Interesse an Service-orientierte Architecturen (SOA) im Bereich Software Entwicklung nimmt stetig zu - vor allem durch die erleichterte Integration von heterogenen Systemen. Web services ermöglichen es komplexe und dynamische Systeme zu implementieren, in denen einzelne Komponenten zur Laufzeit, anhand bestimmter Kriterien, wie z.B. Quality of Service (QoS), ausgewählt werden. QoS-bezogenes Service Management und Service Komposition hängen im großen Maße von diesen wichtigen Daten ab. Viele aktuell publizierte Ansätze gehen jedoch davon aus, dass diese Informationen bereits vorhanden sind.

Diese Diplomarbeit widmet sich diesem fundamentalen Problem und stellt zwei Methoden zur Leistungsüberwachung von WCF basierten Web services, sowie zusammengesetzten Web services, sogenannte Workflows, vor. Die erste Methode basiert auf Windows Performance Counters (WPC), welche von der Windows Communication Foundation (WCF) zur Verfügung gestellt werden. Diese Zähler ermöglichen sehr genaue QoS-Messungen. Die Ergebnisse werden aufgearbeitet und als Metadata in der VRESCo Registry Datenbank gespeichert. Die zweite Methode überwacht Windows Workflow Foundation (WWF) Workflows. Durch das Miteinbeziehen der komplementären Aktivitäten eines Workflows, bekommt man ein differenzierteres Bild über dessen Leistung. Solche Informationen sind ausschlaggebend für QoS Aggregations Algorithmen. Diese Diplomarbeit führt des Weiteren in den State-of-the-Art von Service oriented Architecture ein und gibt einen Überblick über aktuelle Arbeiten in diesem Gebiet.

Contents

List of Figures

List of Tables

Listings

1 Introduction

Information Technology (IT) systems have become key business value enabler. Over the last four decades great efforts have been made to provide the technology and methodology to implement IT systems for small to major companies. Disregarding the size of a company the requirements to its IT infrastructure are almost equal: provide an easy to use and inexpensive mean to master all present and prospective business processes.

One of the latest approaches in this regard is service-oriented architecture (SOA) which represents a new paradigm for the realization and maintenance of business processes. SOA is not just another software development process to implement systems more cost effectively and more efficiently - it aims at providing a maximum of business flexibility. But why is business process flexibility so important? Because flexibility enables a business to respond quickly to a changing competitive environment or to create new business opportunities, thus providing a shorter time to market. Business Process Integration (BPI) is a commonly accepted approach to achieve such flexibility and to improve organizational efficiency. BPI or Business Process Management (BPM) focuses on the effectiveness of end-to-end processes and their desired outcomes. Business Process Management Systems (BPMS) are used to precisely model the enterprise components and possibly change their contexts in which they are used. Processes are defined as a sequential set of activities over their business entities, performed by actors or initiated by events. The complete business structure can be documented, constantly reviewed and improved.

These documented business processes also served and still serve as specification for software implementations. Historical approaches would base their realization on a well designed and highly optimized database schemes. Such a design implicates already the monolithic nature of its resulting application. Changes concerning the database implicated major changes in the entire solution. Thus, former software engineering processes demanded that requirements are known, well defined and fixed at design time. Once the implementation has begun, changes or additional requirements were hardly to accomplish and dedicated project risks. As a consequence changes should

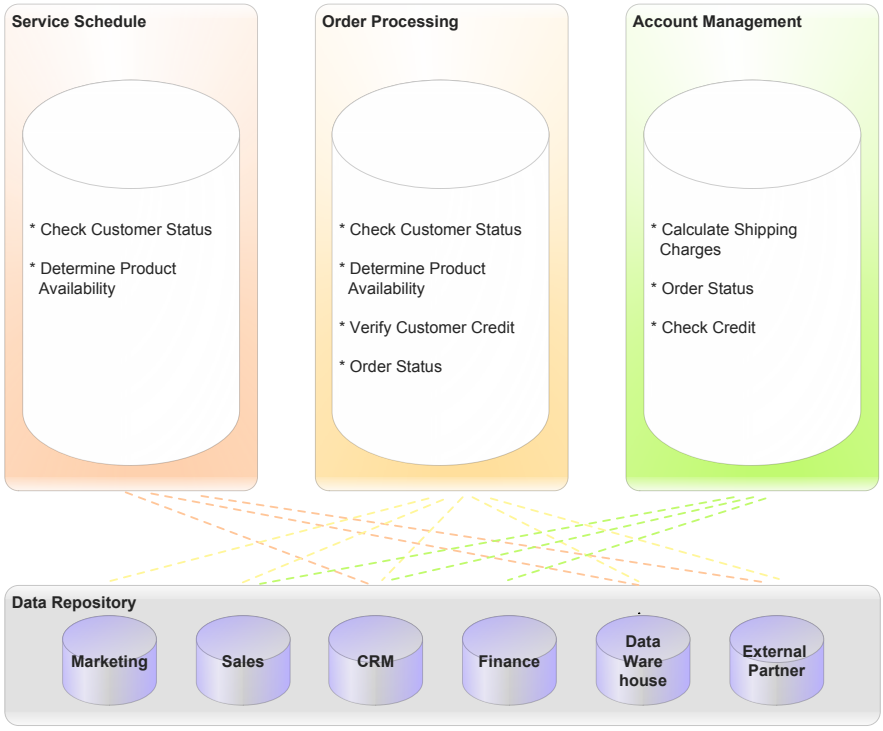

Figure 1: Classical Enterprise Application Architecture

have been avoided or scheduled for the next release. This circumstance certainly degrades business flexibility.
The most serious drawback of such designs is the proceeding divergence of the business process model and its implementation. As a consequence existing software remains in the application infrastructure and is complemented by new systems components.

Example 1. Figure 1 depicts a traditional enterprise application architecture, consisting of multiple monolithic applications. Each of them serves a special purpose: *Account Management* is used to manage the customer relationships. It stores customer information and can check their credit. On the other hand this application is also accessible to the customer to administer his personal data or to check the status of his order. Another application used in this enterprise is responsible for *Order Processing*. Several depart-

ments may use this program to receive informations about order processing or to update the order status. As can be seen by now, the functionality *Order Status* is redundantly implemented in two applications. In order to transfer information from one application to the other, they both have access to the same *data repository* (i.e. *Sales*, *CRM* and *External Partner*). In this example, if a new business process requires an adoption of the functionality *Order Status*, two application have to be changed, tested and rolled out to the enterprise. Apart from the additional expenditure of redundantly implemented functionality, this also degrades the flexibility of the IT infrastructure of this enterprise and consequently the business flexibility of the enterprise itself.

Another problem software engineering has been constantly concerned with, was the disconnect between business users and the IT specialists, which are considered not to speak the same language as their users. Optimized database schemes are difficult to read and understand - even if you are skilled in Entity Relationship (ER) modeling - and system specifications are aimed at aiding the developer in implementing the required functionality. On the other hand, IT professionals often have hardly any knowledge of economics or the domain the software is used in.
By covering several aspects of business process management, service-oriented architecture aims at bridging the gap between software engineers and business analysts. Similar to BPM, SOA decomposes a system into single components and designs them to be reusable in different contexts. XML based technologies like the Business Process Execution Language (BPEL) can be used to define and execute business processes which are an assembly of these components [5]. Therefore software components are published as Web services. This loosely coupled, platform-independent and self-describing components can be composed and orchestrated [27] to implement specified business processes which in turn can be again published as services, called *composite services*. Accordingly, once published the services can be localized and accessed through XML-based standards (i.e. SOAP [31], WSDL [29], UDDI [7]).

Example 2. Example 1 explained a traditional enterprise application which is depicted in Figure 1. Figure 2 shows the same enterprise built as a service-oriented architecture. All business processes remain the same, but they are

realized as Web services. The service layer is depicted as the layer between the Data Repository and the Composite Applications. Web services have direct access to the Data Repository and perform simple and isolated tasks of business processes. These services/tasks are used to compose runnable business processes, which is depicted as the layer beneath the Composite Applications.
As the enterprises business processes remain, consequently the needs to its IT infrastructure do not change. Thus, applications like Account Management and Order Processing are still required. But in a SOA they are built by composing business process from Web services. As depicted in Figure 2 the composite application Order Processing contains the business process Order Processing. This process requires to execute several tasks in a certain order. By arranging selected Web services - also called Web service composition - this business process is declared and added as executable workflow to the composite application order processing.
The mentioned business process Order Processing may require to execute the Web service Check Order Status, which is also required by the process Service Schedule from the composite Application Service Schedule. To implement this business process it only needs to call Check Order Status too. No redundant implementation of this functionality is required.

1.1 Motivation

The introduction already exemplified service-oriented architecture and its reliance on Web services. Hence, a service-oriented design requires the decomposition of business requirements into its mere activities in order to provide loosely coupled and context free components. By providing these attributes Web services can be shared among other software parts, projects or even companies. It is comprehensible that multiple SOA based implementations for enterprises in the same business domain provide many Web services with overlapping or identical functionality.
Thus, SOA's ambition is to reuse existing functionality even if it crosses the enterprises boundaries. Therefore, available Web services should be published using a publicly available service registry. Composite applications are considered to choose among these available Web services the best appropri-

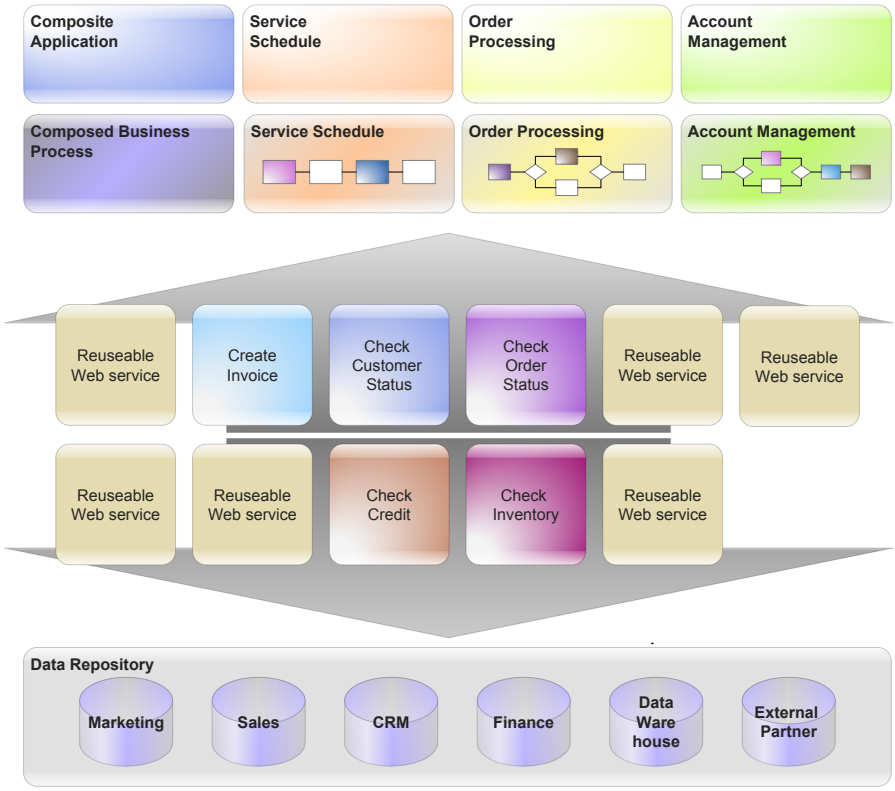

Figure 2: Enterprise Application with SOA

ate one and invoke it. Services should be queried from a registry, bound to it and invoked according to the functional and non-functional requirements of the underlying business process. This is depicted by the SOA triangle (Figure 3).

Nevertheless, as stated by Michlmayr et al. [23] this triangle is broken in certain ways. One major drawback is the inability of dynamic Web service invocation. Currently available solutions (i.e., Apache AXIS [12], Windows Communication Foundation (WCF)), use pre-generated service stubs which are statically compiled and linked to the application. This renders a dynamic invocation impossible since this compiled code is tightly coupled to a certain Web service. Leitner [16] provided a framework to overcome this bottleneck (see Chapter 2.3). Another problem concerning the SOA-Triangle

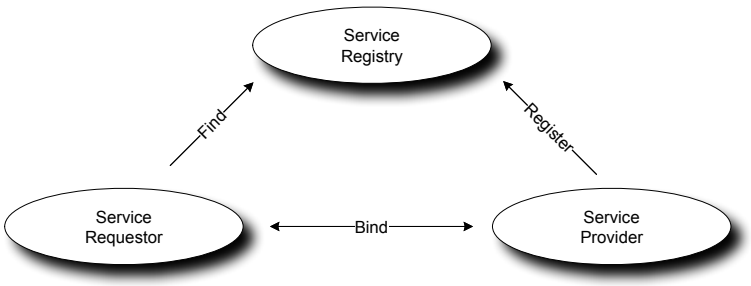

Figure 3: SOA Triangle

are the shortfalls of currently available service registries - mainly the two major solutions UDDI [7] and ebXML [8]. Again, this problem has already been tackled and different solutions have been proposed (see *Related Work* Chapter 3).
Quality of Service (QoS) is one of the main aspects to differentiate certain Web services among a set of semantically equal candidates. The availability of metadata-aware service repositories enables the attachment of QoS data to Web services, which has been extracted i.e., by monitoring and evaluating the performance of a Web service.

One of these QoS monitoring approaches is described in [34] which is part of the Vienna Runtime Environment for Service-Oriented Computing (VRESCo) which is described in more detail in Chapter 2.3. This QoS monitor provides a rich set of well evaluated QoS attributes allowing to independently evaluate and monitor these attributes of Web services. This information is obtained without the knowledge of the service implementation by using low level evaluation of TCP-packets as well as Aspect Oriented Programming (AOP) techniques to invoke the services and extract the QoS values. A part of this thesis can be seen as a direct extension of this work, for submitting its evaluated QoS information to the same QoS management Service, which aggregates it to properly attach it to the external evaluated data.
One drawback of this approach is, that the collected data only represents a client-side view of Web services. Although network delays and communication overhead can be measured, some performance aspects can only be estimated. This QoS monitor could achieve a more accurate result by com-

plementing the the client-side evaluated QoS data with values monitored directly at their hosting system.
The contributions provided by this thesis focus on monitoring Windows Communication Foundation (WCF) Web services as well as Windows Workflow Foundation (WWF) workflows.

From what is known today, no work has been published on observing the Quality of Service of WCF Web services or WWF workflows.

1.2 Problem Definition

In a service-oriented architecture business processes are composed and orchestrated (see Chapter 2.1.2) by the use of diverse tools (see Chapter 2.2). Such a composition can also be referred to as workflow and if the discrete activities of this workflow are Web services with additional QoS related metadata, it can be considered QoS-aware.
The main objective of a QoS-aware workflow is to select services accordingly to fulfill its functional or non-functional requirements. For example, if the workflow ought to be returning a result as fast as possible, Web services with the shortest execution time have to be chosen.

Rosenberg et al. [34] presented an approach to monitoring the QoS of Web services, which has already been introduced as well as its constraints. Further explanations on VRESCo's QoS management can be found in Chapter 2.3.

Chapter 3 will present work related to this thesis, respectively related to QoS-aware service composition as well as QoS monitoring of Web services. The selected papers in this chapter will point out, that most of the currently available literature focuses on new and more efficient QoS aggregation algorithms based on the assumption that accurate QoS data has already been provided.
Approaches which actually focus on the problem of monitoring the performance of Web services can be categorized into server side and client side monitoring. No work so far, has evaluated the monitoring results of both -

client and server side. In the context of QoS-aware service composition such an evaluation could benefit to the efficiency of aggregation algorithms.

This thesis tackles several problems and contributes to this QoS management of the VRESCo runtime by working through the following four steps:

Step 1: Evaluation and Extraction of QoS Information on the Service Host: Solutions presented in [2, 10, 28, 42] monitor the performance of Web services on the provisioning host. But these approaches have to be applied to an application server and can not monitor Web services provided by standalone applications. Further, these prototypes are implemented in the Java programming language. Consequently, they do not integrate properly into the VRESCo runtime which is built upon the Microsoft .Net Framework.

To integrate server side QoS monitoring into the VRESCo environment, Windows Communication Foundation (WCF)[1] based Web services should be constantly monitored on the provisioning host.
Like the client side QoS monitor presented in [34], the server side monitoring should not affect existing or future implementations of the observed services. A solution which has to be integrated into the Web service's source code would require to recompile all services every time, the implementation of the monitor changes. Such an approach is not desirable, considering that in some cases, the provider does not even have access to the source code.
Performance data should be measured constantly and reported to VRESCo in predefined intervals. Some QoS values of a service can be observed passively by listening on events triggered by the Web service during execution. Other parameters have to be actively pulled from the service or sequentially evaluated.
The VRESCo QoS management component has to be extended to aggregate the provided data and associate it directly with services that are managed by the VRESCo environment. This QoS data associated with the corresponding services should be used to calculate the QoS of a workflow by

[1] The Windows Communication Foundation (WCF), provides an API in the .NET Framework for building connected, service-oriented applications.

aggregating the QoS of each atomic Web service.

The QoS data evaluated by this step describes the performance of the server side monitored Web services as well as of the provisioning host. Dynamic service composition based on this data ignores many crucial factors of distributed systems. Thus, also the performance of the client side, which invokes these services, has to be considered. Such a client side monitoring approach is presented in step 3.

Step 2: Using VRESCo Managed Services in Visual Studio Workflow Designer: Since VRESCo provides all concepts needed to dynamically bind to services according to their QoS parameters, it is desirable to use it directly in Visual Studio's Workflow designer. This designer provides tools which allow for creating workflows by dragging several activities onto a worksheet and defining some properties like input/output parameters (workflows and the Visual Studio workflow designer will be explained in detail in Chapter 4.2.1).

Currently, VRESCo services have to be queried an bound to by implementing the required code in the execution-block of a standard `CodeActivity`. This implies a lot of redundant code as well as a precise knowledge of VRESCo's API.

In order to properly evaluate step 1 and 3, an integration of VRESCo managed services is required. The solution should provide the possibility to drag-and-drop activities onto the workflow as well as query for services in the VRESCo registry. When executing a workflow, the activity should bind to the specified service and invoke it dynamically.

Step 3: Evaluating the QoS of Windows Workflow Foundation Workflows: Step 1 provides performance data of Web services which is directly observed on the server. These QoS values are primarily biased by the performance of the server (i.e. hardware, current load conditions, etc.). For effective QoS-aware service composition further QoS parameters have to be taken into account which cannot be gained from measuring the performance

on the server side. Monitoring Web services on the invoking host provides more aspects of the underlying distributed system (i.e. network delays, etc.). In order to evaluate the performance of the workflow QoS aggregation, which is based on data gathered from Step 1, the performance of the executing workflow itself has to be observed.

Two aspects have to be considered when evaluating the performance of a workflow. The first is related to the performance of the invoked Web services. This performance is considered to be directly related to the performance observed on the server side. The second value is retrieved by measuring the performance of the workflow activity, that invokes the Web service on the client side. The difference of these two values represents the overhead of invoking the service (i.e. communication delays, client workload, etc.).
In regard to evaluating dynamic service composition algorithms based on QoS data, it has to be considered that a workflow does not entirely consist of Web service calls. Workflows contain auxiliary activities like `CodeActivities`. Adding such activities to a workflow certainly invalidates a dynamic composition algorithm, which takes only Web service invocations into account.

Therefore an observation method is required, which monitors the QoS of the workflow as well as its distinct activities. The focus of this observation is on monitoring all activities, including peripheral activities like `CodeActivities`, etc.
Again, the monitoring of the workflow should be independent of the workflow's implementation. The observed data should be stored accordingly and made accessible through an API which could be subsequently exposed as Web service in order to integrate into the VRESCo environment.

Step 4: Evaluating the Performance of the QoS Observation Methods: This step performs the evaluation. QoS data gathered from step 1, the server side QoS monitor, is compared to data retrieved from observing WWF workflows. This comparison should be divided into two general evaluations.
The first one should directly compare the server side values with the client side. This could offer valuable clues about interrelationship and predictability of these values.

The second evaluation should compare the estimated QoS of a workflow, based on server side measured performance data, with real performance values, monitored while executing the workflow. This evaluation should also take all peripheral activities into account and determine their deviations.

1.3 Contribution

This thesis contributes basically to the VRESCo environment by introducing the following new components.

WPC-based QoS Monitoring. Step 1 of the problem definition is implemented by using Windows Performance Counters (WPC) of the .Net Framework 3.5. These allow measuring the performance of Windows Communication Foundation (WCF) Web services and provide pretty accurate view on their QoS data.
The monitor is realized as Windows service which can be installed on the service host to enable a loosely coupled integration.

WF-Tracking-based QoS Monitoring. Step 3 is implemented by using the Windows Workflow Foundation (WWF) tracking service. This service can be appended to the workflow runtime and tracks every event initiated by a specified workflow. An evaluation API provides highly accurate execution data, which can be used for workflow composition.

Evaluation Tool. The evaluation tool enables to visualize the performance of of workflow. This performance data can be compared with the values based on the aggregation algorithm. Thus, it aides in algorithm selection and fine tuning.

1.4 Organization of this thesis

The remainder of this thesis is organized as follows:

Chapter 2 details the current state of the art in service-oriented architecture as well as technologies and tools which can be used for its realization. The second part of the chapter will describe the Vienna Runtime Environment for Service Oriented Computing (VRESCo) which this thesis contributes to.

Chapter 3 will give a review of relevant related work in the field of Web service QoS evaluation and QoS-aware Service selection. Additionally these contributions will be evaluated for their applicability within the VRESCo environment and the Windows Workflow Foundation (WWF).

Chapter 4 will explain the realization and implementation of the QoS Monitoring service, the Workflow Tracking and the integration of VRESCo services into Microsoft Visual Studio's Workflow Designer. Furthermore, it will provide some code examples of how to configure and use these services in practice.

Chapter 5 will provide an evaluation of described observation methods. It will compare the QoS data, which is observed by the server side QoS monitor and further aggregated by the VRESCo QoS manager with the data measured by the WF workflow monitor.

Chapter 6 will conclude this thesis with some final remarks and discuss some suggestions for future work.

2 State of the Art Review

This Chapter introduces the concepts and technologies that constitute the state-of-the-art for implementing Service-oriented architectures.

2.1 Service-oriented Architecture

Though SOA is gaining constantly momentum, a global definition is still missing. Currently, the most commonly used definition is stated by the Organization for the Advance of Structured Information Standards (OASIS)[2] as follows:

> **Service-oriented Architecture** (SOA) is a paradigm for organizing and utilizing distributed **capabilities** that may be under the control of different ownership domains. [11]

This very poorly defined and somehow unsatisfying definition led to a variety of misconceptions. Hence, SOA is often reduced to just another software engineering process, which is currently over-hyped. Another misinterpretation is that Web services and SOA are the same thing. A common disbelieve is also that SOA is just adding a Web service wrapper to legacy systems and integrate them into new software - mostly built the same fashion as the old ones.

Service-oriented architecture is not a software engineering process, like the waterfall process model or the unified process, nor is it a concrete tool or framework. It is a new paradigm which provides more flexibility in building business-aligned enterprise applications. By talking of "services" and "processes" it provides a language that is understood by IT specialists as well as business analysts, thus reducing the risk of misconception during analyzing system requirements. Further it is not tied to specific technology, though it highly associated with Web service standards (especially the WS-* standards of the OASIS group).

[2] http://www.oasis-open.org

SOA tackles many traditional problems of software engineering at once. By providing functionality as coarse grained, loosely coupled and especially context free Web services, it makes this functionality interchangeable, reusable and extensible. Therefore SOA defines three distinct roles:

- **Service Provider:** provides Web services and publishes them in in service registries
- **Service Registry:** provides the functionality to register Web services. It acts as broker between the provider and the requesting consumer.
- **Service Requestor:** queries the service registry for Web services, binds to and executes them.

Applied business processes (i.e., by the use of BPEL) use these published Web services to design the business logic accordingly by composing the business inherent processes. Such a composition is also referred to as Web service orchestration (see Chapter 2.1.2). If the business requirements change the affected software is adopted by rearranging the Web services of an orchestrated process. New functionality is once more introduced as Web service, which can be added to a composition.

2.1.1 Web Services

Web services are software components which are made accessible over the Internet.

According to [15] and [33] Web services have to provide the following attributes or characteristics:

- **Self-Contained/Autonomous** - Web services have to be autonomous, so that they can be modified and maintained independently from each other.

- **Coarse-Grained** - Granularity describes the functional richness of a component. Coarse grained Web services provide a higher level of functionality within a single service operation, thus reducing complexity and the number of required service calls.
- **Visible/Discoverable** - Services should be discoverable by the use of a service registry.
- **Stateless** - Service operations do not have a state, nor do they depend on the state or context of any other service.
- **Reusable** - Reusability is intrinsically enabled by applying several other attributes - especially self-containment, loose coupling, coarse granularity, etc. Services can be shared and reused in multiple contexts of different business processes.
- **Composable** - Services can be composed of other services. Complex business processes can be split into several smaller processes which are themselves provided as services.
- **Loose coupling** - Coupling describes the grade of dependability of a component. Loosely coupled Web services have few dependencies which makes them more flexible and maintainable.
- **Self-describing** - The complete description of the Web service is defined by a service contract containing an interface description (operations, input/output parameters, schema, etc.)

Several standards emerged according to Web services: services are described by the Web Service Description Language (WSDL), are published by Universal Description, Discovery and Integration (UDDI), communicate with SOAP and are composed with the Business Process Execution Language (BPEL). Figure 4 depicts the relationship between these technologies. All these standards are XML based and therefore independent from programming languages or providing platforms.

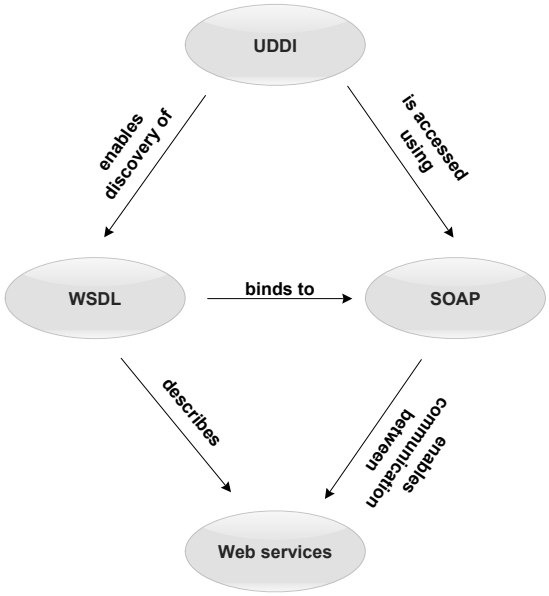

Figure 4: Relationship Between Web Service Technologies [9]

2.1.2 Service Orchestration and Choreography

The arrangement of services to build service-oriented applications can be grouped into two categories: *Orchestration* and *Choreography*.

Orchestration. It describes the composition of executable business processes (see Figure 5) which can interact with internal or widely distributed Web services. By defining a workflow, its execution order and behavior is exactly described. The sequentially execution of these processes is managed by a single controller. BPEL is an example for an orchestration language which can be used for service composition. It provides an orchestration engine that handles the execution of the composed processes as well as the invocation of Web services.

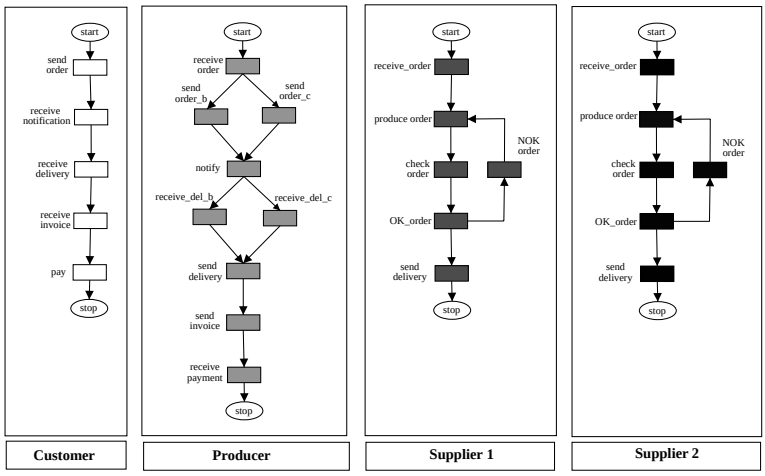

Figure 5: Orchestration [6]

Choreography. It focuses on describing the observable behavior of the system from a global point of view. It is intended for long-running multi-party collaborations. It provides a decentralized solution where each party has knowledge of their part of the process and which also accepts that control can be distributed among the different parties.
Figure 6 depicts an example choreography with four different participants. Each party represents a separate company providing services which are consumed by others.

Choreography techniques are more complex to model but offer a decentralized alternative to classic orchestration models, where all data passes through a centralized orchestration engine, which results in unnecessary data transfer and wasted bandwidth. Thus, the engine becoming a bottleneck to the execution of an orchestrated workflow, choreography permits a peer-to-peer approach.

A commonly used analogy compares orchestration and choreography with a dancing couple. Thus, orchestration describes a single dancer, which knows the dancing steps (which are predefined rules), whereas choreography describes how the two dancers act as a pair and how the dance is performed.

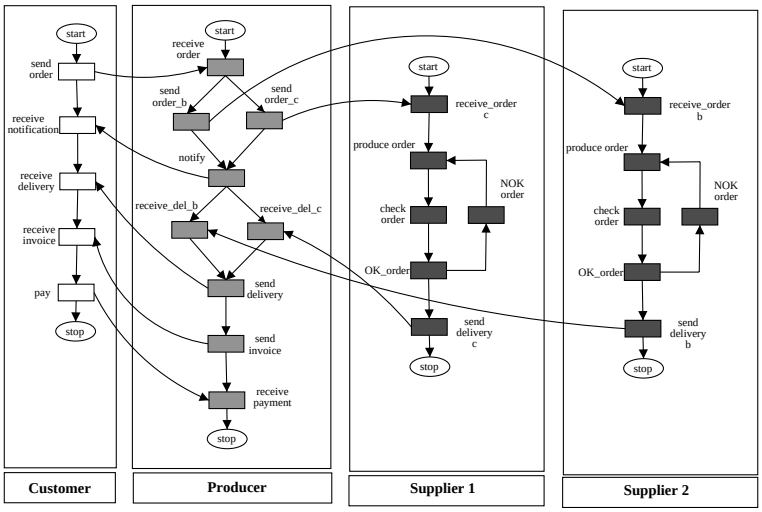

Figure 6: Choreography [6]

2.2 Tools and Technologies

In accordance with the rise of Web services many technology standards have populated and further contributed to their success. Figure 7 shows a Web service standards stack, depicting the relationships between those technologies.

2.2.1 WSDL

The Web Services Description Language (WSDL) [29] is a XML based document format for precisely defining Web services. This important standard is developed and maintained by the World Wide Web Consortium (W3C)[3]. WSDL service descriptions are divided into three layers:

- The interface description of a Web service - declaring all operations with input and output parameters as well as their types.

[3]http://www.w3.org

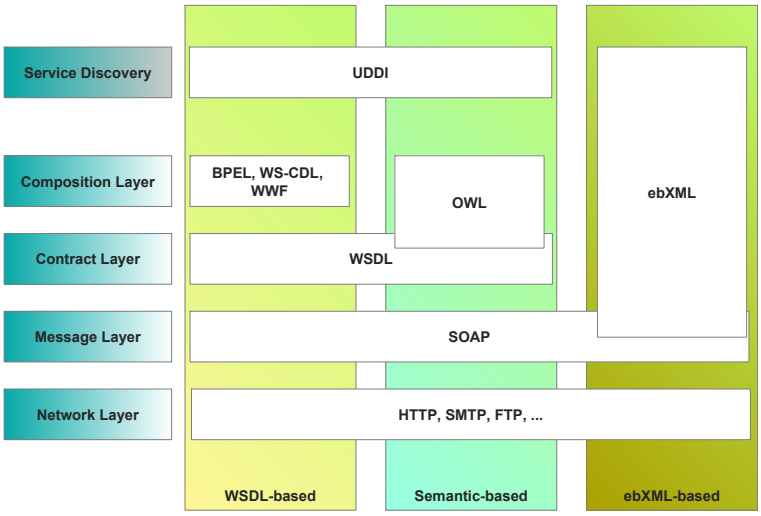

Figure 7: Web Service Standards Stack

- The binding of a Web service - describing the protocol format for which it is available.
- The location of a Web service - describes the physical location (URL) where the service is hosted.

The general structure of a WSDL file is depicted by Figure 8.

WSDL is generally useful regarding code generators. JAX-WS[4] for example, generates RPC-Style service Stubs for the Java programming language. Microsoft Visual Studio also provides simple to use code generators to use Web services with their programming languages. The other way around, WSDL files are usually not implemented manually, but created by automated tools.

2.2.2 SOAP

SOAP originally was an acronym for Simple Object Access Protocol, but this acronym was dropped in Version 1.2 for not being simple and not dealing

[4] http://jax-ws.dev.java.net/

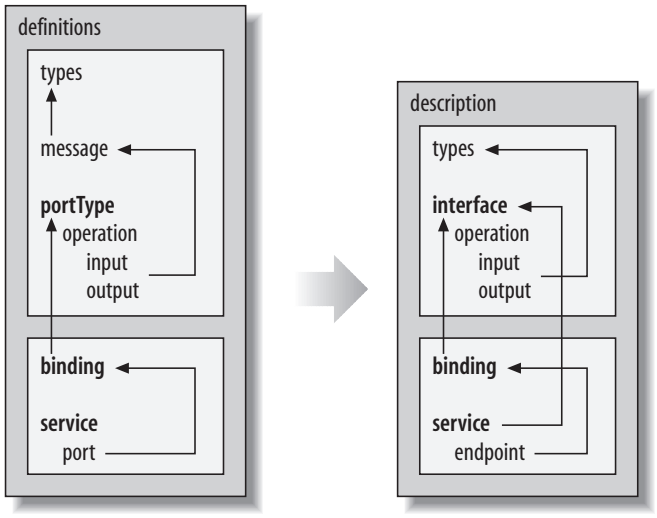

Figure 8: General Structure of WSDL 1.1 and 2.0 [15]

with objects - now it is just a name. It is a XML based communication protocol for exchanging encoded data over networks. SOAP can utilize a variety of of protocols (i.e. HTTP, SMTP, MIME) as transport layer but the binding over HTTP is universally used on the Internet.

SOAP is independent of any programming model and supports one-way messaging and various request-response type exchanges including RPC. It has widely been accepted as the De facto standard for Web service communication, which is emphasized by countless implementations in several programming languages.

2.2.3 UDDI

The Universal Description, Discovery and Integration (UDDI) is like nearly every Web service standard, a XML based standard for Web service registries. It is developed and maintained by OASIS and is intended to expose information about one or more businesses and its provided Web service interfaces.

Registries can be run on multiple sites and are accessible by everyone or every program connected to the Internet, providing information about service providers, service implementations, and service metadata. UDDI intends, that service providers register their services in a registry, which can be accessed by consumers to query for them. The registry acts like the yellow pages - an application in need of a particular service can search for it by using the registry, request its location, bind to it and invoke it. This is also referred to as the SOA triangle (see Figure 3 in Chapter 1.1).

Initially UDDI was intended to be even more than a simple lookup service. It was dedicated to form a worldwide UDDI Business Registry (UBR) which should have been an universal central broker for all Web services worldwide. Unfortunately UDDI did not succeed as intended and Microsoft, SAP, and IBM have finally shut down their public UDDI registries in January 2006 [24]. But instead of admitting the failure of UBR, its initial motivation of a global yellow pages for Web services was redefined to always have been intended a prototype:

> This goal was met and far exceeded. The UBR ran for 5 years, demonstrating live, industrial strength UDDI implementations managing over 50,000 replicated entries. The practical demonstration provided by the UBR helped in the ratification of UDDI specifications as OASIS standards... [24]

2.2.4 ebXML

Electronic Business using eXtensible Markup Language (ebXML) is a family of XML based standards initiated by OASIS providing standard methods for exchanging business messages or define and register business processes. This suite of specifications initially intended to provide standards for business processes, core data components, collaboration protocol agreements, messaging, registries and repositories. Finally five specifications were submitted and approved by the International Organization for Standardization (ISO), thus representing the legitimate ISO standard 15000:

- ISO 15000-1: ebXML Collaborative Partner Profile Agreement
- ISO 15000-2: ebXML Messaging Service Specification
- ISO 15000-3: ebXML Registry Information Model
- ISO 15000-4: ebXML Registry Services Specification
- ISO 15000-5: ebXML Core Components Technical Specification

2.2.5 WS-CDL

Web Services Choreography Description Language (WS-CDL) [30] is a Web service specification developed by the W3C WS-CDL Working Group, in order to provide peer-to-peer collaborations for participants from different parties. These collaborations between the interacting participants are defined from a global point of view.
WS-CDL offers a fully expressive global description language based on XML.

2.2.6 WWF

The Windows Workflow Foundation (WWF) provides a programming model, a workflow engine and additional tools for building workflows and workflow-enabled applications which can be designed in Visual Studio .Net 2005 and 2008, up to now.

A workflow is a set of Activities which are arranged to model a real process (i.e. business process). This activities are executed sequentially or according to their states either by system functions or human interaction. The WWF runtime engine resides in-process and provides the base activity libraries, the runtime engine as well as runtime services (i.e. Persistence Service, WWF Tracking service - chapter 4.3.2) and can host multiple Workflows at once.

2.2.7 BPEL

The Business Process Execution Language (BPEL) is a widely adopted industry standard for orchestrating Web services. BPEL is used to abstract business logic by assembling Web services to build "composite" services. These services are again deployed as Web services and can be integrated into other orchestrations.
BPEL is a XML based language without a graphical representation, though several implementations have introduced visual workflow designers. But the lacking definition of a representational standard lead to inhomogeneous use of existing standards (i.e. the Business Process Modeling Language (BPML)).

BPEL was initiated as a joint effort by BEA, Microsoft, and IBM but is now maintained by the Organization for the Advancement of Structured Information Standards (OASIS) as an open standard.

Though BPEL was originally also initiated by Microsoft, Windows Workflow Foundation (WWF) workflows does not adhere to this standard. There is only an add on WWF in the .NET Framework 3.0 providing import and export functionalities.

2.2.8 OWL

Web Ontology Language (OWL) developed by the Web Ontology Working Group[5] of W3C an ontology language for the Semantic Web. Ontologies describe the meaning of terminology used in Web documents by defining a set of formalized vocabularies of terms and their relationships among each other.
It enables applications to process and understand the content of information instead of just representing it. Thus, it makes Web content more readable and interpretable for machines and replacing simple keyword search by content based queries.

[5] http://www.w3.org/2001/sw/WebOnt/

OWL provides three sublanguages. Each of it with increasing expressiveness:

- **OWL Lite** - supports only a simple classification hierarchy and simple constraints. It is less complex than OWL DL and is aimed to be simple to adopt for tool providers.
- **OWL DL** - supports maximum expressiveness and includes all OWL language constructs (with some restrictions).
- **OWL Full** - supports maximum expressiveness and the syntactic freedom of the Resource Description Framework (RDF), but is does not give computational guarantees. It can be seen as an extension of RDF.

2.3 VRESCo

The VRESCo project (Vienna Runtime Environment for Service oriented Computing) aims at solving some of the major shortcomings in Service Oriented Computing (SOC) [25]. The project was introduced in [23] and addresses the initial idea of the SOA triangle to publish-find-bind-execute a service.

VRESCo targets multiple topics among which are the following:

Service Discovery and Metadata. As already stated, a substantial problem with today's SOA concerns the shortcomings of current Web service registries such as Universal Description Discovery and Integration (UDDI) [7] or ebXML [8]. These two standardized registries only provide keyword-based query of services and do not attach metadata or non-functional properties to their records, reducing these commonly as "the standards" recognized registries to simple lookup services. Thus, leading to commonly scratching service registries from service-centric system designs and abandoning the initial SOA concept. This is underlined by IBM's, Microsoft's, and SAP's decision to shut their public UDDI registries down underlines this in 2005 [24].
Service metadata gives additionally information about the service's behavior, which can't be distinguished by service description languages such as the Web Services Description Language (WSDL) [29].
Once annotated with additional metadata services cannot simply be looked up, but be queried by certain attributes such as QoS or pre- and post-conditions.

Dynamic Binding and Invocation. Referring once again at the basic principle of SOA, one of the main concepts is to dynamically bind and invoke services from a pool of relevant candidates. Currently this is only possible if the service interfaces are identical. Assuming that services should be interchangeable irrespective their location or provider, it is optimistic to expect all of them being designed the same way by different people, respectively to

implement the same interface.
Leitner et al. [16] describes several requirements for Web service invocation frameworks that support the core SOA ideas. Among these are *stubless service invocation* to decouple services from pre-generated stubs, *protocol independency* to abstract from underlying Web service protocols like SOAP- and REST. A *message driven* approach should be emphasized compared to an RPC style, which leads again to tighter coupling. *Asynchronous (nonblocking) communication* is required to support processes with long execution times. And at least an *acceptable runtime behaviour* is expected.

Service Versioning. Extending or simply changing an interface in an traditional programming language results in the need to recompile the complete project - or at least the depending modules. This also represents the current state of the art of changing Web service interfaces. For client-side Web service invocation it is common to use generated client-stubs, which have to be regenerated every time the Web service's interface changes. Compiling and packaging these stubs with the application, binds them as well as the application, to the Web service's version being available at design time.
This is again a problem of dynamic binding and invocation. A different version of a service is equivalent to a different implementation of another provider.
VRESCo supports service versioning by abstracting services into service revisions which can define successor-predecessor relationships between different versions, thus mapping a revision-graph similar to conventional versioning systems like CVS[6] or SVN[7].

2.3.1 VRESCo Architecture

The overall architecture of VRESCo is shown in Figure 9.

[6] Concurrent Version System [http://www.cvshome.org/]
[7] Subversion version control system [http://subversion.tigris.org/]

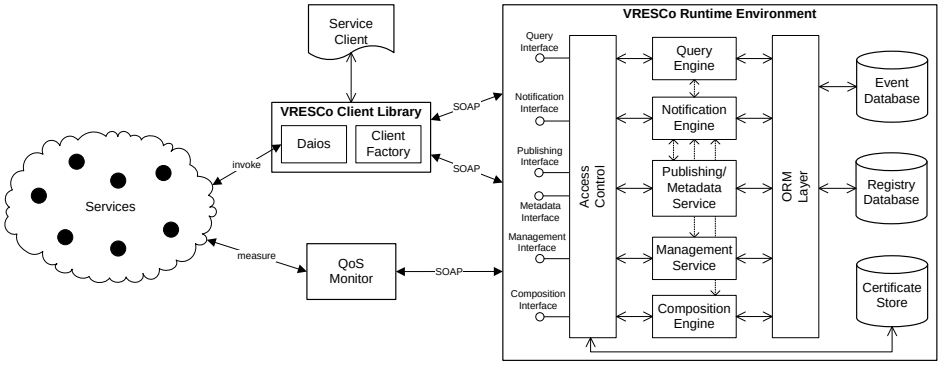

Figure 9: VRESCo Architecture Overview [22]

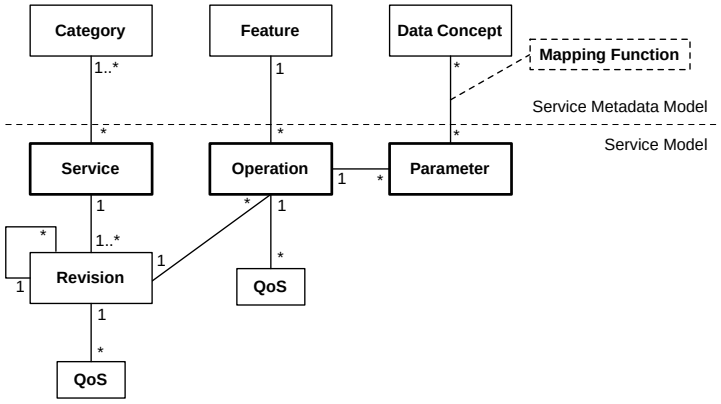

Figure 10: Service Model to Metadata Mapping [20]

Publishing & Metadata Interface. This service is used to publish services with its interface description and associated metadata into the registry. This can be invoked dynamically at runtime or statically using a Web-GUI. Figure 10 depicts how services and metadata are mapped accordingly.

Querying Interface. The querying service allows to query for available services that have been published to the registry. A special querying language - the *VRESCo Query Language* (VQL) - provides the functionality to query generically and type-safe all information stored in the database.

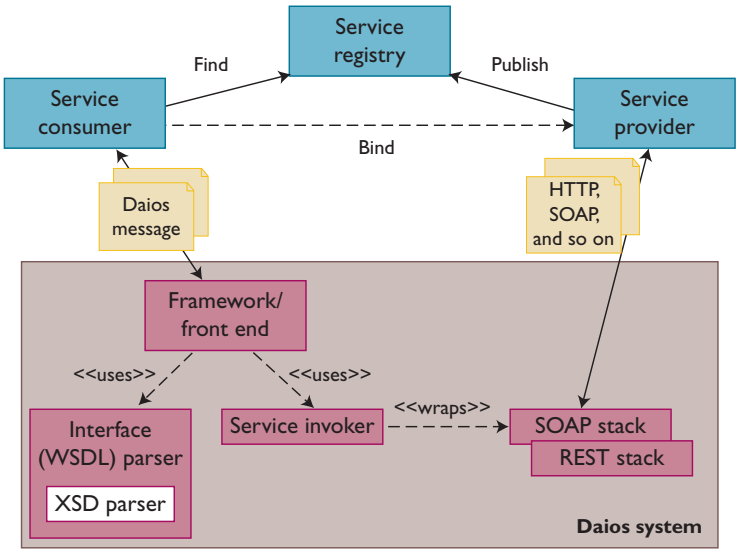

Figure 11: The Daios Framework Architecture [16]

VQL's API is similar to Hibernate Criteria API [32], but does not provide a declarative querying language like SQL.

Binding & Invocation Interface. Binding to services is managed through the Daios framework [16, 17]. This framework supports all requirements initially explained in the preceding chapter. Figure 11 shows the Daios framework's general architecture and its accordance with the SOA triangle (see Figure 3 on page 16).

Table 1 summarizes the currently supported rebinding strategies.

QoS Monitor. The QoS Monitor evaluates and monitors performance related QoS attributes of Web services. This is achieved independently of the service implementation and the providing platform, thus it can be applied to every running Web service without interfering or even knowing its implementation. The QoS Monitor uses object-oriented and aspect-oriented programming as well as low level TCP-Packet evaluation methods to re-

Strategy	Description	Cost
`Fixed`	Rebinding is not needed.	none
`Periodic`	Proxy verifies rebinding periodically.	Constant overhead
`OnDemand`	Proxy rebinds on request of the client.	low overhead; binding not always accurate
`OnInvocation`	Proxy rebinds in advance of every service invocation.	accurate bindings; degrades service invocation time
`OnEvent`	Uses VRESCo notification engine to rebind on event notifications defined by users.	related to defined situations.

Table 1: VRESCo Rebinding Strategies

trieve the QoS values. Services are constantly monitored and the resulting information is stored as metadata in the VRESCo Metatdata Registry. The observed data represents a client side view of a monitored Web service.

Composition Service. The composition service [35] aims at providing an end-to-end system for QoS-aware service composition. A a domain-specific language - the Vienna Composition Language (VCL) - based on the Microsoft Oslo Framework[8] is provided, which is able to specify constraints describing functional and non-functional aspects of a composition. It also enables *Composition as a Service* (CaaS)

Notification Interface. The Event Notification Interface [21] publishes notifications within the runtime on the occurrence of certain events (i.e. service removed, user added, etc.). Additionally to standard notification, the VRESCo interface also publishes QoS related events, as well as information about binding, invocation or runtime. Clients can subscribe to the interface and get notified about events according to their predefined rulesets.

[8] http://www.microsoft.com/NET/Oslo.aspx

3 Related Work

Huang et al. [14] presented a related framework for dynamically invoking Web services in a workflow by using a proxy service. An optimization service selects, according to QoS data provided by a data collection service, an appropriate Web service from a set of semantically similar services. This approach mainly focuses on dynamic invocation. The QoS aggregation is merely reduced to sequential samples of CPU-load averages. These averages are set in relation to the CPU-speed according the formula:

$$\frac{CPUspeed}{LoadAverage + 1}$$

The use of proxy services provides flexibility and enables dynamic invocation of Web services, but also adds an additional layer to the invocation chain, which impacts the overall performance.
The QoS model used by this work is not very significant. On the contrary, this thesis aims more distinctly at the extraction of several QoS attributes and contributes to the VRESCo runtime. WPC-based QoS monitoring and the WWF tracking service, as proposed in this thesis, are executed separately (on different host, if desired) to the Web service execution and at a very low level of either the operating system or the Web server. Thus, the additional overhead is minimal.

Zhang et al. [40] extend OWL-S [36] with a lightweight QoS model and provide QoS driven service selection in dynamic composition of Web services. The focus of this work is on the optimization of service selection algorithms by categorizing the QoS constraints. For acquiring the necessary QoS information the authors refer to [38], which a middleware platform that enables quality-driven composition of Web services.
The QoS model is similar to the model used in VRESCo. This work mainly concentrates on local and global workflow optimization according to QoS attributes which can be weighted through user defined rules. Nevertheless, the paper leaves the question, how the QoS data is acquired, unanswered.

Zulkernine and Martin [42] present a framework for monitoring and verifying

Service Layer Agreements (SLA) of composite Web services-based processes. This framework contributes to their Comprehensive Service Management Middleware (CSMM) [41] which facilitates the collected data to QoS-based service discovery, SLA negotiations and workflow orchestration and execution. The performance monitor verifies that a preliminary defined workflow satisfies a set of negotiated SLAs. It is realized as message interceptor on the SOAP message processing layer of the server that hosts the Web service. The presented performance monitor requires an application server to install the message interceptors. Thus, Web services which are provided by standalone applications cannot be monitored. WPC-based QoS monitoring, as introduced by this thesis, can be applied to all Web services, that are based on the .NET platform.

Fei et al. [10] present a policy-driven monitoring framework for collecting QoS information and adapting service provisioning for cross-domain service interaction. The framework is built on their distributed QoS registry *Q-Peer* [18], which is a P2P (Peer-to-Peer) QoS registry architecture for Web services. Services are observed by capturing SOAP messages. The monitor collects data, generates metrics and gives feedback in case of QoS deviations. Monitoring policies can be submitted at runtime as well as different monitoring models.
Though, this framework thoroughly prepares evaluated data to be stored in the QoS registry, the evaluation method itself is similar to [42] and thus, requires the services to run on an application server.

Raimondi et al. [28] introduced online monitoring of Web service SLAs based on timed automata [1]. Message handlers are injected into the open-source AXIS[9] engine from Apache. These handlers are installed on the server side and monitor the hosted services. So called checkers are added to the client side and provide information about invocation performance. A handler is a auto-generated Java based timed automata. SLAs are seen as timed words which can be generated by these automatas. If the handler produces a word, that is not in accordance with the SLA, a violation has occurred.
Like the tools presented in this thesis, their approach is also non-intrusive and can be used on existing Web services without altering their implementa-

[9] http://ws.apache.org/axis/ - Web services - Axis

tion. It also uses server and client side monitoring to gather a comprehensive set of performance data.

Artaiam and Senivongse [2] propose a new QoS model for Web services which extends the commonly used parameters (e.g. response time, successful execution rate, availability) by some additional parameters (e.g. security and regulatory). The paper also provides a metric for each quality attribute from service providers' perspective. The implementation of Web service monitoring is based on Sun's Java application server *Glassfish*[10]. A managed bean, based on the Java Management Extension (JMX)[11], is deployed, which wraps the access to a Web services and captures QoS data. Additionally the existing service endpoint AggregateStats is extended, which provides access to monitored statistical data gathered from managed beans. A security analyzer discovers relevant vulnerabilities of Microsoft Windows hosts.

Sun et al. [37] present a prototype framework for monitoring BPEL-based Web service compositions. Aspect Oriented Programming (AOP) is used to extend the open-source BPEL engine ActiveBPEL[12] with monitoring logic. This logic is derived from Web service policies which are used to generate the AOP code. Once the logic is added, it produces monitoring information which is further processed and analyzed using Extended Message Sequence Charts (EMSC).
The presented solution presents client side workflow monitoring. The methods used to observe QoS information, are similar to the those used by the VRESCo QoS monitor [34], which also uses AOP and other techniques to extract performance data from running Web services. It also provides an analyzing tool. This thesis provides workflow tracking for the .NET platform and validates the client side observed performance against highly accurate server side QoS data.

Zeng et al. [39] developed a QoS observation metamodel. The paper states, that it is not practical to provide a predefined set of QoS metrics due to the broad concept of QoS and its domain specific contexts. The metamodel can be used to construct various QoS observation models which fit for the certain

[10] https://glassfish.dev.java.net/
[11] http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement/
[12] http://www.activebpel.org/

domains of the projects. This approach uses service operational events to calculate the metrics. It also uses model driven techniques for QoS modeling and hybrid compilation to generate the monitoring code.
This approach also uses operational events which are produced by the surrounding application server during the Web service execution.

Baresi et al. [3] present two methods of monitoring Web services of a BPEL process. This approach uses annotations to comment designed processes with contracts on the desired behavior. Contrary to the other presented approaches it only accounts for timeouts, runtime errors and violations of the functional contracts. Data is gathered from the BPEL process itself by adding fault handlers to its implementation. A second approach uses the xlinkit[13] rule and validation engine to verify correct invocations of Web services.
By using annotations, the presented approach needs to be part of the implementation and cannot be applied to existing Web services like the monitors presented in this thesis. The second approach which uses the xlinkit validation engine implicates considerably XML-processing. The WPC monitoring solution, proposed by this thesis, uses the integrated Windows performance counters, which are built into the operating system and thus, representing a lightweight solution to performance measuring.

[13]http://www.systemwire.com

4 Design and Implementation

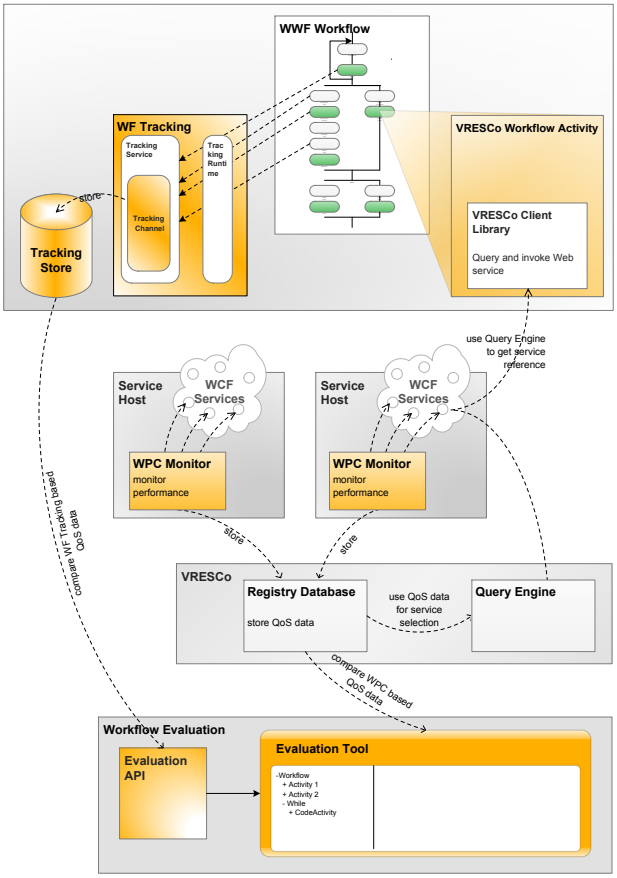

Figure 12: Solution Overview

The following Chapter will detail the architecture of the provided solution as well as the implementation of each of its components. Figure 12 describes how these components work together. The implementation is mainly structured into four parts. Each part realizes a specific step of the problem definition in Chapter 1.

To efficiently monitor and evaluate the performance of a Windows Workflow Foundation workflow two values have to be compared. The first one is the server side performance of the component Web services. This ser-

vices are usually scattered upon several hosts. To monitor the Web service's performance, each provisioning host has to run a properly configured WPC performance monitor (see Chapter 4.1). These monitors evaluate the QoS of the services and transmit their results sequentially to a VRESCo runtime environment.
To use VRESCo managed Web services in workflow compositions, the `VRESCo-WebserviceActivity` is used, which queries VRESCo for service revisions, binds to them and invokes them when the workflow is executed. The Web service provided by VRESCo is queried based on the QoS data retrieved by the QoS monitor as well as the WPC performance monitor (see Chapter 4.2).
A WWF workflow triggers several events while executing. These events can be processed by the WWF tracking service. A customized WWF tracking service is used to track all relevant workflow events and evaluate the client side QoS of the monitored Web services as well as the performance of the entire workflow (see Chapter 4.3). The results of this performance evaluation is stored in the tracking store.
Finally an evaluation API and an evaluation tool is provided in Chapter 4.4, which can be used to evaluate the QoS values retrieved by the WPC performance monitor and the WWF tracking service.

4.1 WPC-based QoS Monitoring of Web Services

This chapter describes the realization of step 1 - the Windows Performance Counter (WPC) based QoS Monitoring of WCF Web services as described in the problem definition (see Chapter 1.2). This implementation contributes to the VRESCo environment by transmitting evaluated QoS data to the framework, which is then stored in its metadata repository.

4.1.1 Overview

Dynamic service composition is highly dependent on accurate QoS data of its component services. Chapter 3 summarized several approaches of monitoring the QoS of Web services and discussed their shortcomings. Chapter 1.2 defined several problems and subdivided them into four solvable steps.

WPC-based QoS monitoring tackles the first step - the server side QoS monitoring of Web services. Server side monitoring represents the performance measurement of Web services on their provisioning host. When realized properly, this implies very accurate and reliable results, because the number of sources influencing the QoS of a service are limited. The performance of the client computer, network delays, communication overhead, etc. have not to be considered. Furthermore, these QoS parameters are already evaluated by VRESCo's QoS monitor [34]. Adding server side QoS evaluation to the already existing QoS monitoring facilities of the VRESCo environment, provides a more complete picture of the overall QoS of a SOA. It also allows for a more difficile dynamic service composition. By knowing exactly the performance of a Web service on the provisioning host, it is possible to derive other important QoS attributes (i.e. network delays) instead of estimating them.

Windows Performance Counters. The Microsoft Windows operating systems provides an extraordinary simple and lightweight tooling for evaluating the performance of processes, resources, etc. - the Windows Performance Counters (WPC). These performance counters are a monitoring and

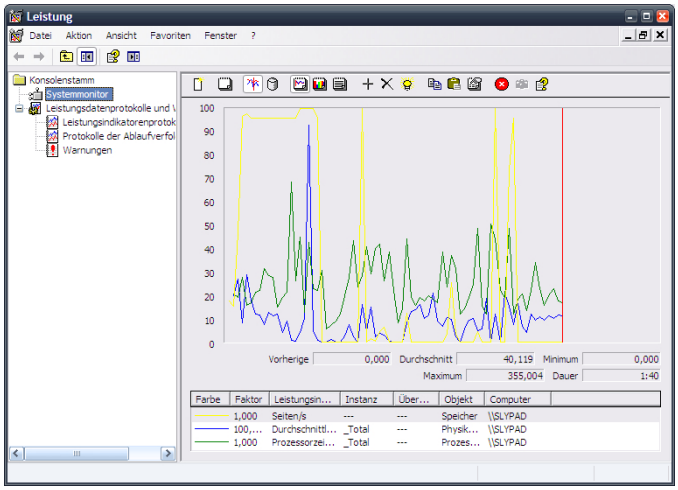

Figure 13: Windows Performance Monitor

analysis instrument at the management instrument layer of the Windows Communication Foundation, which is deeply integrated into the operating system. The implementation of these counters is as simple as it gets, which is one of their main advantages.

The Windows platform declares dozens of performance counters. The most commonly known is `% Processor Time`, which is used by the Windows Task Manager to display the activities of running processes. Example B describes how performance counters are used in another performance evaluation tool – the Windows Performance Monitor.

The Windows Communication Foundation (WCF) introduces three additional categories of performance counters - `ServiceModelService` for monitoring parameters related to the application domain and the service host, `ServiceModelEndpoint` for monitoring Web service endpoints and `ServiceModelOperation` for monitoring the discrete operations - each category defining and implementing a set of counters to analyze the performance of a hosted WCF Web service. A complete overview of all WCF performance counters is provided in Appendix 3.

Example 3. The Windows Performance Monitor (see Figure 13) is a tool provided by the Windows operation system, which can be used to diagnose performance problems. A set of performance counters can be selected which

are sequentially polled by the Performance Monitor and displayed as line charts. By default the monitor reads every second the current values of the selected counters and updates the chart. This provides a real time graphical display of the results which can be used to identify performance bottlenecks and allows for a system administrator to take action.

The reason why this solution uses WPCs is, that they provide a highly accurate means of monitoring the QoS of a Web service. Furthermore, this can be achieved without altering any existing or prospective services. By taking advantage of the three performance counter categories, introduced by the WCF, it is possible to track Web services through examining these counters. Another advantage of WPCs is, that they permanently keep on counting - even if the QoS monitor is not running. This is just because the counters are managed by the operating system, independently from the QoS monitor. If the WPC-based QoS monitor crashed or is disabled for a certain timespan, the performance counters remain. The QoS for this timespan can be evaluated, the next time the WPC-based QoS monitor is started.

4.1.2 Architecture

The WPC-based monitoring service is realized as Windows service. This architectural approach provides advantages like automatic startup at boot time. Once installed, a Windows service can be configured, started or stopped from the *Services Control Manager* .
This service - the WPC-based QoS monitor - has to be installed on each host providing a Web service, that should be monitored. Nevertheless, as depicted in Figure 12 it is not necessary that these services all remain on the same host, nor that the WPC-based QoS monitor remains on the same host of the VRESCo runtime. The WPC-based QoS monitor uses VRESCo's client library to transmit the evaluated QoS data to the VRESCo runtime.

The second architectural decision was, to use the performance counters provided by the Windows Communication Foundation [19, 26]. It requires only an additional parameter in the Web services `App.config` or `Web.config` file, to advice Windows to attach the performance counters to the service.

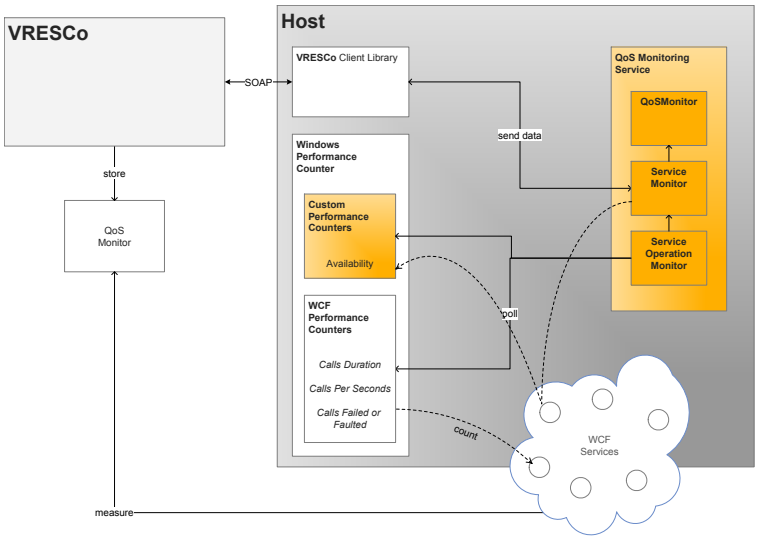

Figure 14: QoS Monitoring Service Architecture

Adding additional parameters to an already existing configuration file, cannot be considered as altering the implementation. Thus, the WPC-based QoS monitor can be applied to every WPC Web service, disregarding its implementation.

The format of the data retrieved by the performance counters is not conform with the QoS model used by VRESCo. Thus, the according data is mapped to apply this format (see Chapter 4.1.3). After the measured results have been aggregated and mapped accordingly, they are transmitted to VRESCo's QoS Management Service. The data is queued as QoS-Event and has to be processed by VRESCo's QoS Scheduler. This scheduler aggregates the values of the WPC-based QoS monitor and stores them in the metadata repository.

4.1.3 Quality of Service Model

For being an Add-On to VRESCo's QoS Monitoring Service, the Quality of Service Model of the WPC-based QoS monitor is adopted from [34].

VRESCo QoS Param	Perfmon	Description
`ExecutionTime`	**Calls Duration** (WCF)	The timespan from invoking an operation to its termination, aggregated as mean value over the measuring interval (milliseconds)
`Throughput`	**Calls Per Seconds** (WCF)	The number of calls to an operation per second aggregated as mean value over the measuring interval
`Accuracy`	**Calls Failed or Faluted** (WCF)	The accuracy of this operation (in percent) over the measuring interval
`Service Available`	**Availability** (not WCF)	The availability of the Web service (in percent) over the measuring interval

Table 2: QoS Monitoring - Mapping of Performance Counters to VRESCo QoS Parameters

ExecutionTime. The execution time $t_e(S, o)$ of an operation o of a Web service S is the mean duration this operation needs to execute (measured in milliseconds). This mean value is aggregated over the measuring interval[14]. The utilized performance counter "Calls Duration" of the counter category "ServiceModelOperation 3.0.0.0" already calculates a mean value respectively to the last time it has been read. This implies that the proceeding value is reseted, which can be resulting inaccurate samples for this attribute, if other applications but the WPC-base QoS monitor invoke this performance counter.
To render this problem the provided aggregation methods are dispensed and another low level approach has been taken to retrieve this QoS parameter appropriately. Technically this performance counter sums-up the call duration of each execution of the assigned operation as well as the total number of executions. Additionally, it also counts how often this operation has been executed. Respectively, represents $\sum_{i=0}^{n} e_{wpc}$ the number of all executions of the operation since the initialization of the Web service until the sample has been read.
To calculate this value, two samples of the performance counter have to be

[14]The measuring interval is a configuration parameter of the WPC-base QoS monitor and is explained in detail in Table 3.

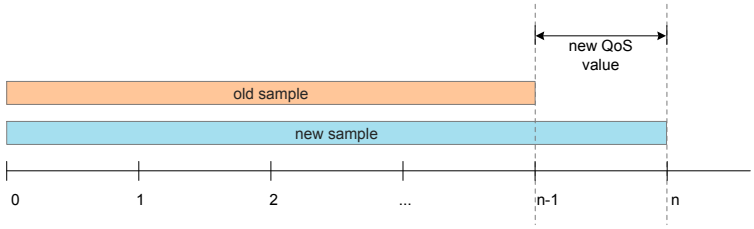

Figure 15: QoS Calculation by Subtracting Overlapping Intervals

taken (see Figure 15). Each sample taken of the performance counter represents the total sum of all "Calls Duration" (respectively "execution time") since the initiation of the Web service on the corresponding host. Thus, the *new sample* of Figure 15 has a higher "Calls Duration"-value $\sum_{i=0}^{n} t_{e_{wpc}}$ than the *old sample* $\sum_{i=0}^{n-1} t_{e_{wpc}}$. This is because the number of executions of the current sample $\sum_{i=0}^{n} e_{wpc}$ is higher than the number of the old sample $\sum_{i=0}^{n-1} e_{wpc}$.

Accordingly, the desired execution time t_e of a service's S operation o is calculated as the difference between the sum of call durations from the last and the current samples, divided through the difference of total calls. In the following equation as well as in the equations of the other QoS parameters, n represents the current sample number and $\sum_{i=0}^{n} t_{e_{wpc}}$ describes the sum of all execution times stored by the performance counter up to the n-th sample.

$$t_e(S,o) = \frac{\sum_{i=0}^{n} t_{e_{wpc}} - \sum_{i=0}^{n-1} t_{e_{wpc}}}{\sum_{i=0}^{n} e_{wpc} - \sum_{i=0}^{n-1} e_{wpc}}$$

Throughput. Throughput $tp(S,o)$ of an operation o of a Web service S represents the number of requests that can be processed within the monitoring interval (measured in executions per minute). The assigned performance counter "Calls Per Second" cps_{wpc} of the counter family "ServiceModelOperation 3.0.0.0" is of the same type as the one used for "Calls Duration". Therefore, it is again calculated as the quotient of the difference between the number of total requests per second cps_{wpc} from the last sample and the

current sample and the difference of the total requests e_{wpc} between the last and current sample.

$$tp(S,o) = \frac{\sum_{i=0}^{n} cps_{wpc} - \sum_{i=0}^{n-1} cps_{wpc}}{\sum_{i=0}^{n} e_{wpc} - \sum_{i=0}^{n-1} e_{wpc}}$$

Accuracy. The Accuracy $ac(S,o)$ of an operation o of a Web service S is defined as the success rate of o as the relation between successful and failed operation requests (measured in the number of failed requests per second). It is assessed by utilizing the performance counter "Calls Failed Per Second" $cfps_{wpc}$ of the counter family "ServiceModelOperation 3.0.0.0". This value is calculated exactly as the Throughput - the quotient of the difference between the number of total requests per second $cfps_{wpc}$ from the last sample and the current sample and the difference of the total requests e_{wpc} between the last and current sample.

$$ac(S,o) = \frac{\sum_{i=0}^{n} cfps_{wpc} - \sum_{i=0}^{n-1} cfps_{wpc}}{\sum_{i=0}^{n} e_{wpc} - \sum_{i=0}^{n-1} e_{wpc}}$$

Service Available. The availability $av(S)$ of a Web service S is the probability that this service is up and running and producing correct results over the period of the monitoring interval (measured as percentage over the monitoring interval). The data is retrieved from a custom performance counter as described later on page 56 and is calculated as the quotient of the difference between the number of successful availability-check requests sr_{wpc} from the last sample and the current sample and the difference of the total availability check requests r_{wpc} between the last and current sample.

$$av(S) = \frac{\sum_{i=0}^{n} sr_{wpc} - \sum_{i=0}^{n-1} sr_{wpc}}{\sum_{i=0}^{n} r_{wpc} - \sum_{i=0}^{n-1} r_{wpc}}$$

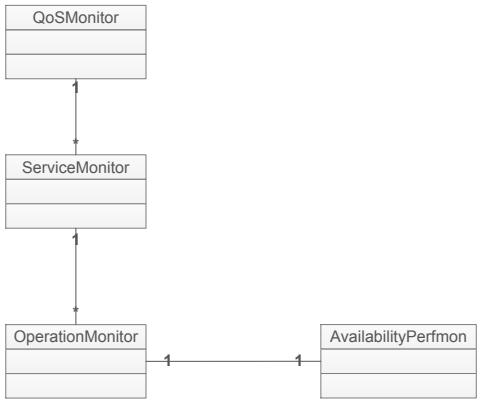

Figure 16: WPC QoS Monitor Class Diagram

4.1.4 Implementation

The WPC-based QoS monitoring service has been implemented solely using the .NET Framework 3.5.
All main functions are implemented by four classes (see Figure 16). The component is represented by the `QoSMonitor`, which is responsible for loading the settings from the configuration file, mapping the performance counters to VRESCo managed Web services and providing an API to control the monitor. The Windows service uses this API to initialize, start and stop the `QoSMonitor`.

Usage of WCF Performance Counters in the QoS Monitoring Service. Performance counters are instantiated and constantly polled. Some data can be mapped directly to VRESCo QoS parameters, some has to be aggregated to match its representative data format. Table 2 shows which performance counters are used and how they are mapped to VRESCo QoS metadata attributes. A complete overview of available performance counters in the Windows Communication Foundation is listed in appendix B

Mapping Performance Counter Instances to Web Services and Operations. The unique identifier, also called `InstanceName`, of a WCF per-

formance counter of the category "ServiceModelService 3.0.0.0" is a character string built of two parts: the name of the Web service and the service endpoint. If the resulting length of the string exceeds a certain length, the components of the identifier will be truncated and replaced by hex-values before they are concatenated.

Example 4. A Web service called "BookFlight" is provided with the service endpoint address "http://localhost:8011/bookingprovider/bookingservice". The corresponding `InstanceName` would be:

```
bookflight@http:||localhost:8011|bookingprovider|bookingservice
```

The VRESCo publishing service is hosted on "vresco.vitalab.tuwien.ac.at" with the service name "publishingservice"' with the service endpoint "http://-vresco.vitalab.tuwien.ac.at:20000/publishingservice". This would result in the following identifier:

```
publishingservice@http:||03ac.at:20000|publishingservice
```

The mapping of Web services to performance counters is realized through string matching. For each Web service specified in the QoS monitors's `App.Config` file, a `ServiceMonitor` object is created and the appropriate `ServiceRevision` is queried from VRESCo. The `ServiceRevision` provides all relevant informations about this Web service. With this information provided the `QoSMonitor` iterates through all `InstanceNames` and checks if they contain the name of the service.

The mapping of a Web service to an instance of the performance counter category "ServiceModelService 3.0.0.0" is necessary to successfully map the Web service's operations to their corresponding performance counters. For each Web service all instances of the category "ServiceModelOperation 3.0.0.0" are retrieved. Again, the names of the operations is provided by the `Service-Revision`. The identifier of a `ServiceModelOperation` performance counter is similarly assembled as an `InstanceName` of a `ServiceModelService` counter. Every identifier contains the name of the operation as well as the name of the corresponding Web service. This simplifies the mapping of a service to all its operation performance counter instances. For each operation, an

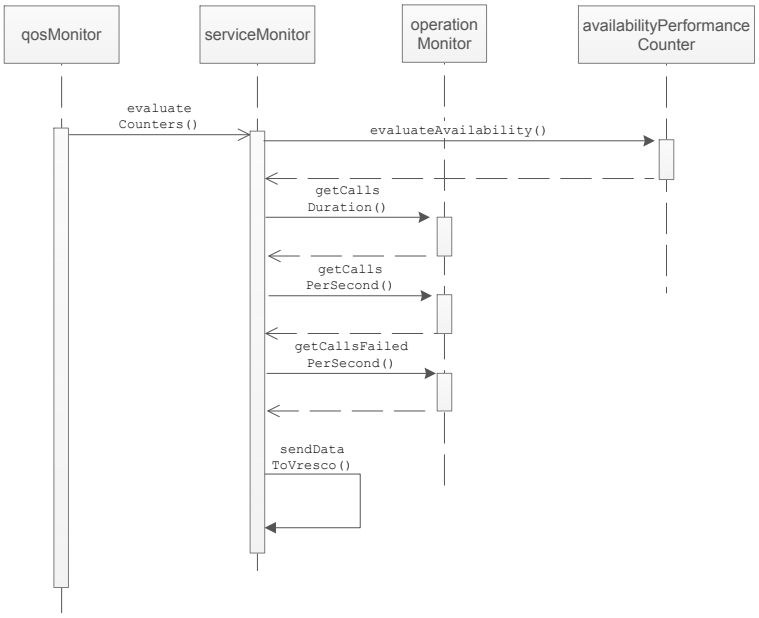

Figure 17: QoS Monitoring Service SequenceDiagram

`ServiceOperationMonitor` object is created and added to the corresponding `ServiceMonitor`.

Passive Monitoring. After a successfully mapping of Web services and their operations, the `QoSMonitor` contains a set of `ServiceMonitors`, which in turn contain a set of `ServiceOperationMonitors` (see Figure 14). The latter implement the access methods to the performance counters as well as the QoS calculations based on the formulas of Chapter 4.1.3.
Figure 17 depicts the procedure of QoS information retrieval. The `QoS-Monitor` is the controlling instance and sequentially triggers all its associated `ServiceMonitos` to start their performance measuring. The `Service-Monitor` collects the required QoS data for each operation from its corresponding `ServiceOperationMonitor` and sends this information to VRESCo. As denoted in Table 2, all WCF performance counters (*ExecutionTime*, *Throughput* and *Accuracy*) can be monitored passively, because the `Service-OperationMonitor` only needs to poll the values of these counters, that are

increased on the occurrence of events triggered during the execution of the according Web service.

Active Monitoring - Monitoring Availability. Chapter 4.1.3 stated that *"the availability of a Web service is the probability that this service is up and running and producing correct results over the period of the monitoring interval"*. This QoS parameter cannot be measured passively. If a service is not invoked for a certain time, the corresponding throughput decreases - but, the only way to assess the availability of a Web service is to actively invoke it. Thus, the accounting of availability information is impossible due to the design of Windows performance counters, because it would require them to actively invoke themselves to observe the presence of a service.
Accordingly a custom performance counter has been implemented. The `QoSMonitor` sequentially triggers the `ServiceMonitors` to check the availability of their services. The `ServiceMonitor` tries to poll sample data from a performance counter instance. This instance is only available if the appropriate service is currently running. If this procedure succeeded, the availability counter as well as the number of availability checks is increased. If the trial failed, only the number of checks is increased. Thus, the availability is the relation of successful trials to the over-all number of trials.
The length of the interval of the `AvailabilityCheck` can be declared by specifying the parameter `availabilitycheckinterval` in the `App.Config` file (see table 3). The shorter the interval the more accurate the aggregated value of availability, but the higher the additional of the CPU.

4.1.5 Installation and Configuration

The WPC QoS monitoring service has to be installed on the host, which provides the Web services. Being part of the VRESCo framework, it is obvious that these services have to be successfully registered with VRESCo runtime, though it is not necessary for the runtime to reside on the same host.

Option	Description
General Configuration Options	
`monitoringinterval`	Specifies the measuring interval (in milliseconds)
`availabilitycheckinterval`	Specifies the interval of the availability check (in milliseconds)
`webservices`	Specifies one or more webservices that should be monitored
Web service Configuration Options	
`wsdl`	Specifies an URL to the wsdl-file
`revisionid`	Specifies the RevisionID of this service
`operations`	Specifies one or more operations to be monitored. If no operations are supplied, all operations of this service will be monitored
Operation Configuration Options	
`name`	Specifies the name of the operation

Table 3: QoS Monitoring - Configuration Options

Installing and Running the QoS Web Service Monitoring Service. To install the WPC QoS monitoring service a separate MSI-Installer has been created. After the successful installation of the service, it has to be configured accordingly. This is done by setting the required parameters in the `App.Config` file, which is located in the installation directory.
The service can be started and stopped by the Windows service interface. Once started the service constantly continuous to monitor the specified services.
If the service does not find a running VRESCo runtime on its initialization, it suspends for 10 minutes and tries again. Starting the specified Web services after the QoS monitor does not affect its behavior and only results in a poor availability.

Configuring the QoS Web Service Monitor. The QoS Web service Monitor can be configured by setting its parameters in the `App.Config` file. This file needs to include the newly implemented `vresco.qosmonitoring` configuration section at the top (see Listing 2).

The configuration section is exampled in Listing 3 and has to supply the already explained parameters of Table 3. Each Web service has to specify the URL of its describing WSDL file. Additionally the according VRESCo `RevisionID` can be supplied (see Web service configuration options, Table 3). If only certain operations should be monitored, each operation has to be supplied. Only these operations will be monitored and the others are ignored. If no operation is supplied, the QoS monitoring service will automatically observe all operations of this Web service. Service and operation names are queried case-sensitive by VRESCo's querying service.
Finally performance counting has to be enabled in the hosting application's `App.Config` file by adding the parameters of Listing 1.

```
<diagnostics performanceCounters="All"  wmiProviderEnabled="true" />
```

Listing 1: QoS Monitoring - Enable performance counters

```
<configSections>
  <section name="vresco.qosmonitoring"
           type="QoSMonitoringService.QoSMonitoringConfig,
                 QoSMonitoringService,
                 Version=1.0.0.0,
                 Culture=neutral,
                 PublicKeyToken=null" />
</configSections>
```

Listing 2: QoS Monitoring - Configuration Section

```
<vresco.qosmonitoring monitoringinterval="20000"
                      availabilitycheckinterval="5000">

   <webservices>
      <webservice
         wsdl="http://localhost:8011/SMSProvider/A1SMSService?wsdl">
      </webservice>
      <webservice
         wsdl="http://localhost:8013/SMSProvider/OneSMSService?wsdl"
         revisionid=1 >
         <operations>
            <add name="SendSMS" />
         </operations>
      </webservice>
   </webservices>

</vresco.qosmonitoring>
```

Listing 3: QoS Monitoring - Example configuration

4.2 VRESCo Integration into WWF Designer

This Chapter describes the implementation of step 2 of the problem definition (see Chapter 1.2) - the integration of VRESCo managed Web services into the Visual Studio workflow designer.

4.2.1 Overview

This task preliminary contributes to the next task, which introduces a new method of QoS-measuring of Windows Workflow Foundation (WWF) [4] workflows (see Chapter 4.3). This chapter will provide a short overview of workflows and how this thesis contributes to

Workflow. A *workflow* describes a sequence of activities, agents and dependencies between activities which are necessary to reach a predefined goal - generally, workflows are used to model real work or small decomposed patterns of it. They are less specific than a process description - for not including well-defined inputs, outputs and purposes. Agents can be humans or other interacting components like software systems. The dependencies determine the execution sequence of activities, which can be executed sequentially, repeatedly in a loop or in parallel.
The typical representation of a workflow is a directed graph where nodes correspond to activities and edges correspond to activities (see Figure 18) and are designed with graphical modeling tools.

Microsoft's Visual Studio also integrates a workflow designer since version Visual Studio 2005. It is part of the WWF and can be used to create workflow templates by simply dragging activities from the toolbox onto a

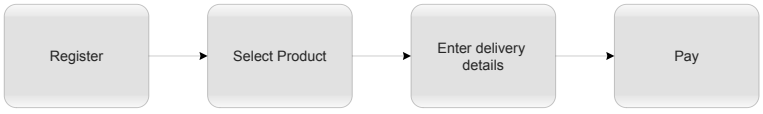

Figure 18: Example Workflow - Simplified Online Shop

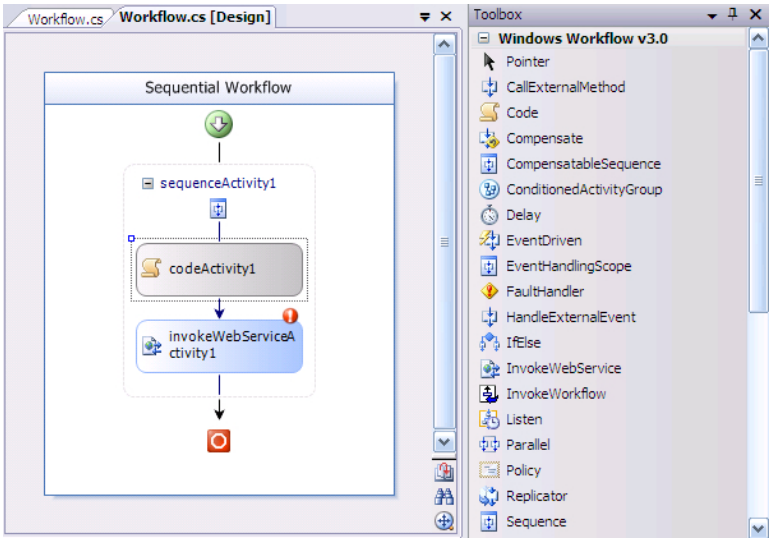

Figure 19: Visual Studio Workflow Designer

workflow template (see Figure 19). The designer integrates very well into the development environment and provides plenty of convenient tools as well as full debug capabilities.

WWF workflows are very well suited for Web service composition. A rich toolset of activities are provided to control the execution of a workflow. To invoke a Web service, service references can be bound to the workflow and `InvokeWebService`-activities are used, to call the operations of these services. This binding of services during the design of the workflow, presents some comforts, like automatic import of operations and their parameters, etc. On the other hand, this statically binds a certain Web service to a workflow. To change this service, the workflow has to be recompiled.
Dynamic Web service composition requires Web services to be assigned at runtime - while the workflow is executed, or at least during its initialization. The VRESCo environment provides facilities for dynamic service composition, but a direct integration of VRESCo into Visual Studio has not been implemented, yet.

Though the WWF designer provides proprietary activities for Web service invocation, the only way to invoke a service which is registered at

VRESCo currently is, to query, bind to and execute it within a standard `CodeActivity`. This very cumbersome approach requires the developer to know exactly specific information of the designated Web service. The vast amount of redundant code, which is increased with each additional service call represents another drawback of this approach. Thus, a custom activity is implemented to drag-and-drop a VRESCo Web service activity right into a visually designed workflow in Visual Studio. This activity only needs some parameters to identify the requested service and takes care of binding and invocation procedures.

The solution provided by this thesis consists of two activities, the `VRESCoWebserviceActivity` and the `VRESCoRebindingActivity`. The `VRESCoWebserviceActivity` can be used, to call VRESCo managed Web services with fixed rebinding. The `VRESCoRebindingActivity` uses a query to select the Web service that should be invoked.

4.2.2 VRESCoWebserviceActivity Implementation

The `VRESCoWebserviceActivity` is derived from `System.Workflow.ComponentModel.Activity`, entailing it with the basic activity properties and functionality.

The design decision to provide the activity as linkable library imposes the need to specify the path to the project's `App.Conifg` file on each instantiated `VRESCoWebserviceActivity`. This prerequisite emerges from the VRESCo Client Library which is utilized by the activity to query the registry for Web services. The client library depends on the VRESCo configuration properties which are declared in the `App.Config` file. This configuration file resides in the project's workspace environment. The activity, however, is injected into Visual Studio's platform environment. Access to the workspace environments of the separate projects is only provided to Visual Studio plugins.

The implementation overrides the Execute method to perform the required actions of binding and invoking the specified Web service.
The activity binds to a service using VRESCo's client library. Because it

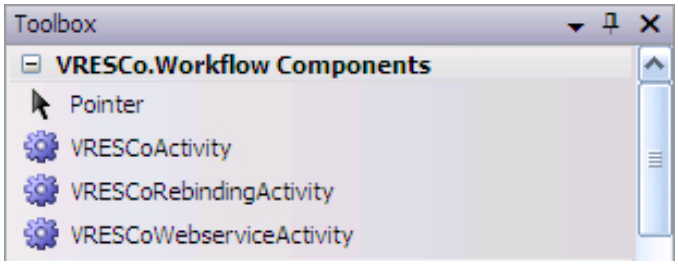

Figure 20: VRESCo Activities in Toolbox

binds to a specific service revision, the `RebindingStrategy` is always fixed. This revision is retrieved from VRESCo Querier and a `RebindingProxy` is instantiated and stored statically in the `LocalQuerier`. This `LocalQuerier` represents the layer between the VRESCo client library and the `VRESCoWebserviceAcitivity`, as well as the `VRESCoRebindingActivity` and encapsulates all access to the client library. Furthermore, it creates the rebinding proxies (see Chapter 2.3) for each VRESCo activity and stores their references statically. Thus, the proxies remain and can be referenced by the next workflow invocation. This is required to enable rebinding.
The specified Web service is then invoked using the supplied `RequestMessage`. According to the specified `RequestPattern` the service response is stored in the `ReceiveMessage` or discarded.

4.2.3 Using the VRESCoWebserviceActivity

In order to use the `VRESCoWebserviceActivity` the library as well as the referenced VRESCo libraries have to be included and referenced in the project. After successfully including the required libraries, the *VRESCo.Workflow Components* category containing the `VRESCoWebserviceActivity` is added to the Toolbox (Figure 20). The activity can be applied by simply dragging it onto the workflow.

First of all, when using the activity, the path to the project's `App.Config` file has to be supplied. Thus, the configuration has to contain all attributes that are necessary to access the VRESCo client library. This declaration is only needed to provide the Workflow designer with proper information, but

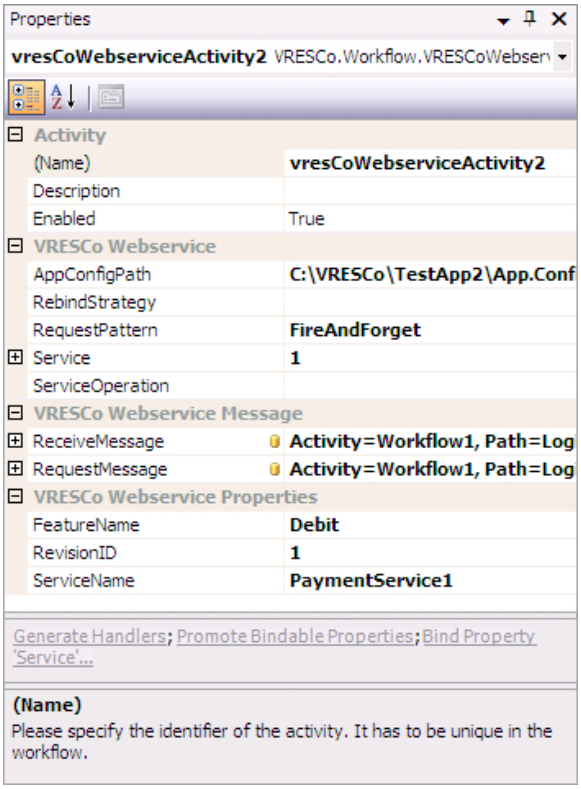

Figure 21: VRESCoWebserviceActivity - Properties Dialog

does not affect the runtime behavior of the activity.

Figure 20 shows the available configuration options of a `VRESCoWebservice-Activity`. The parameters are described in Table 4. The first two parameters are standard attributes derived from the WWF activity. An activity has to be uniquely named and can be provided with an additional description. The `RequestPattern` defines how the Web service should be invoked. *RequestResponse* calls the operation and waits for the result message. *Fire-AndForget* instead just invokes and ignores the result.

To specify a Web service the according `RevisionID` can be entered into the properties dialog. This will trigger the implementation to query VRESCo for the corresponding Revision and to supply all necessary attributes with the retrieved data. Accordingly the *Revision Search Dialog* proceeds after

Property	Description
`Name`	Unique name of the activity
`Description`	Additional description of the activity (optional)
`Enabled`	Enable/Disable the activity
`AppConfigPath`	Absolute path to the project's `App.config` file containing the URLs to the VRESCo core services
`RequestPattern`	The required request pattern (with or without response)
`Service`	The `RevisionID` of the VRESCo managed service (can be entered directly or selected by using the search-form
`ServiceOperation`	Specifies the required Operation
`RequestMessage`	The `DaiosMessage` for the service request
`ReceiveMessage`	The `DaiosMessage` for the service result

Table 4: VRESCoWebserviceActivity - Properties

selecting an adequate Revision. The *Revision Search Dialog* is described beneath. After successfully selecting the `ServiceRevision`, the Operation-`DropdownList` of the properties dialog will be updated with all operations that have been published to the VRESCo registry.

The required `RequestMessage` has to be globally accessible in the workflow and has to be built in advance by using the constructor of the workflow or by prepending a *CodeActivity* to the `VRESCoWebserviceActivity`. `Request-Message` and `ReceiveMessage` are realized as *DependencyProperties*, allowing them to be selected through a special dialog. This dialog can be opened by clicking on the dotted button in the properties field.

The VRESCo Revision Search Dialog. The Revision Search Dialog (see Figure 22) is a convenient way to query a service that is managed by VRESCo. It allows to search for Web services according their Feature- or Operation-name. The search is wild carded, thus, the exact name of a feature or operation is not required.

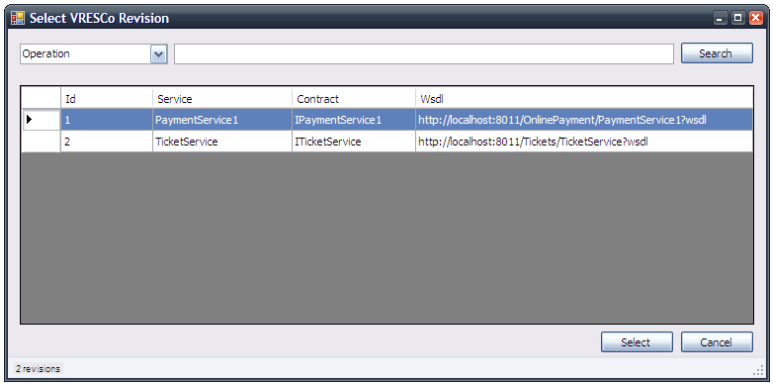

Figure 22: VRESCoWebserviceActivity - Choose Revision

4.2.4 VRESCoRebindingActivity Implementation

The `VRESCoRebindingActivity` is intended to enable dynamic Web service composition. By defining a query, Web services can be selected according certain parameters (i.e. feature name, QoS, etc.). This query is passed on to the `LocalQuerier`, which creates a `RebindingProxy` based on these parameters. Thus, the `VRESCoRebindingActivity` itself has no direct reference to a Web service, but instead it has a description of what the service should do and how it should perform. The service is queried during the execution of the workflow and the proxy binds to it.

This activity also uses the `LocalQuerier` to access proxies and execute queries. Furthermore, a query builder has been implemented, which allows for building and testing queries. These queries are stored as a CSV-strings and could also be entered without the query builder.

4.2.5 Using the VRESCoRebindingActivity

Again, the activity requires the path to the project's `App.config` file, to provide the configuration for the client library. The activity has to be dragged onto the workflow and the properties of Figure 23 have to be set.

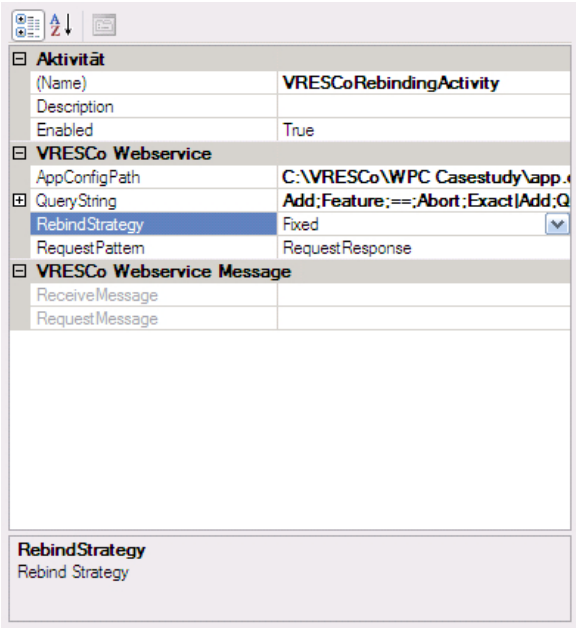

Figure 23: VRESCoRebindingActivity - Properties

The `QueryString` can be built manually or by using the query builder (see Figure 24). To build the string manually, the query has to be formatted in the following sequence: *Add/Match ; Search Criterion ; Comparator ; Search Value ; QueryMode*
Add/Match specifies the VRESCo expression type. A *Search Criterion* is any kind of search criterion supported by VRESCo's query engine (i.e. Feature, QoS, etc.). The *Comparators* are those of standard programming languages (i.e. Java, C++, etc.). The *Search Value* is any string, which does not contain a semicolon. The `QueryMode` is a string representation of VRESCo's `QueryMode` and accepts the values `Exact`, `Priority` and `Relaxed`. If more then one parameter is entered, the parameters have to be separated by "`|`".

Example 5. A query for Web services, which implement the feature "Abort" and have a response time of less then 400 milliseconds could be manually provided by the following CSV-formatted string:

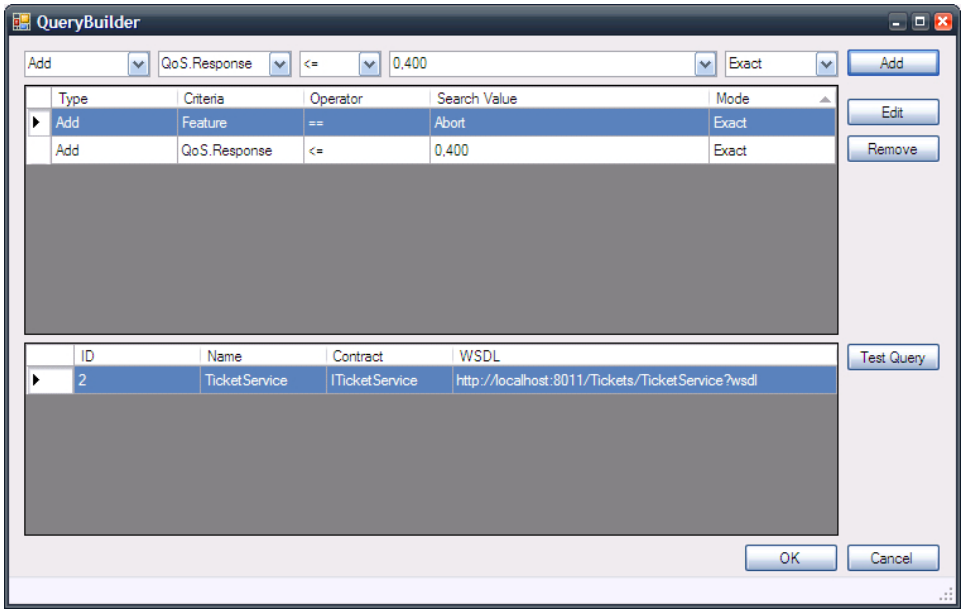

Figure 24: VRESCoRebindingActivity - Query Builder

```
Add;Feature;==;Exact|Add;QoS.Response;<=;0.400;Exact
```

This query can also be entered by using the query builder. The described example is depicted in Figure 24.

The Query Builder Dialog. The query builder (see Figure 24) provides a Graphical User Interface (GUI) to easily build and test queries. The query is built like described in the previous paragraph. The first `ComboBox` defines the VRESCo expression type. The second selects the search criterion, with several predefined values. It uses the `LocalQuerier` to retrieve some information from VRESCo. For example, if the search criterion "Feature" is selected, the search value's `ComboBox` is updated with all features, currently registered at VRESCo. In this case, the `Search Value` can be selected using the `DropdownList` of the `ComboBox`. If another criterion is selected, this value can also be entered manually.

By pressing "*Add*", the settings of all `ComboBoxes` are stored in a new row

of the query table below. Each row represents an expression of the query and can be edited or deleted. This query can also be tested by pressing "*Test Query*", which builds a query of all entered expressions and executes it. The results are displayed in the second table of the query builder.

The query is stored by pressing "*Ok*". This closes the query builder and stores the query as formatted string in the properties field `QueryString`.

4.3 WWF Workflow Monitoring

The following chapter describes the monitoring of Windows Workflow Foundation (WWF) workflows which is realized by utilizing the WWF tracking service. It implements the requirements as described in step 3 of the problem definition (see Chapter 1.2).

4.3.1 Overview

Chapter 4.1 described the WPC-based QoS monitor as an approach to solve step 1 of the problem definition (see Chapter 1.2). This solution represents server side monitoring of the QoS of a Web service. A workflow, as described in Chapter 4.2.1, is composed of several Web services. During its execution, all specified services are invoked in sequence. Thus, a workflow represents a Web service client as well as the monitoring of a workflow represents the client side observation of QoS values.

Differences of client and server side monitoring have already been mentioned in the problem definition. Server side monitoring is expected to be very accurate, because the performance of a Web service is only influenced by a few parameters. On the other hand, QoS values observed at the client side, have to be differentiatedly analyzed, because there have to be much more components taken into account (i.e. the underlying network, the performance of the client host, etc.).
Some of this influencing factors are already analyzed be the QoS monitor introduced in [34] (see also Chapter 2.3). This monitor provides additional network related information like latency, response time and round trip time. Combining the results of WPC-based QoS monitoring with network related information observed by VRESCo's QoS monitor, QoS aggregation algorithms, like those presented in Table 30 on page 93, are expected to be reasonably accurate.

Aggregating the QoS of a workflow only by the QoS of its component services disregards the fact, that a workflow may contain several auxiliary activities. The Windows Workflow Foundation (WWF) provides a set of activities,

which either call external methods or can be used to implement the executable code. Current literature has not addressed this problem, yet. The solution described in this chapter provides a tool to monitor the performance of a workflow. Furthermore, it allows for tracking the performance of each of its component activities as well. This provides valuable insights in the overall performance of a workflow and which allows for optimizing them. The data can also be used to tweak QoS aggregation or dynamic composition algorithms.
This information is also evaluated by this thesis. Chapter 4.4 introduces an evaluation tool, which allows for clicking through the activities of a workflow. Chapter 5 provides a structured evaluation by comparing the estimated performance with observations by this workflow monitor.

Chapter 4.1 introduced Windows performance counters as a highly accurate measuring tool. Unfortunately the Windows Workflow Foundation does not provide additional counters which could be used to track the performance of workflows. Though, WWF provides an internal service to monitor workflows, the WWF tracking service, which will be explained in detail in Chapter 4.3.2.

4.3.2 Architecture

The architecture of the WWF Workflow Monitoring is built upon the WWF tracking service. This service provides means to track events that are initiated at several execution points of activities during the execution of a workflow. It is part of the WWF workflow runtime and is only triggered when a Workflow is executed.

WWF Tracking Service. The WWF Tracking Service [13] can be used to track workflows. An executing workflow sends events which are categorized by the following event types:

- *Workflow Events* occur when the state of a specific Workflow changes (Created, Completed, Idle, Suspended, Resumed, Persisted, Unloaded,

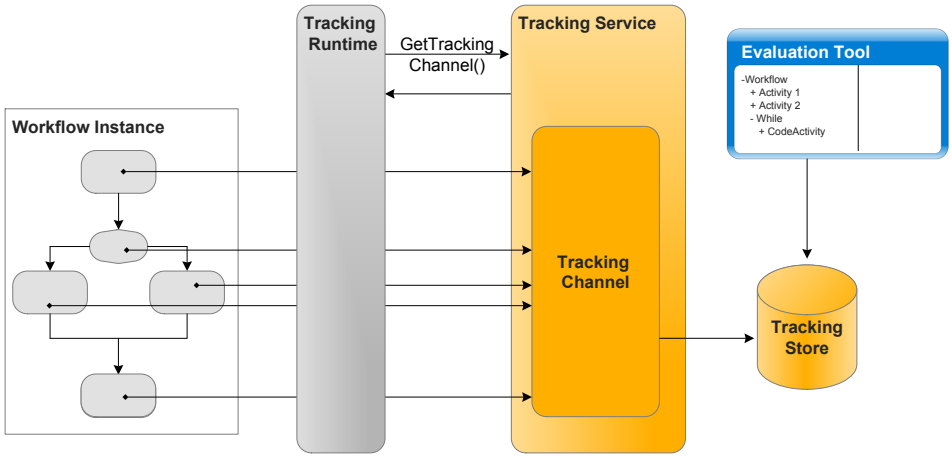

Figure 25: Windows Workflow Tracking - Architecture

Loaded, Exception, Terminated, Aborted, Changed, Started)

- *Activity-Execution Status Events* are similar to workflow events and pronounce the the currently entered state of the concerning activity (Executing, Closed, Compensating, Faulting, Canceling).
- *User Events* are custom events generated by the application. These events enable to pass relevant application-specific information to the tracking service which in turn can be observed by other applications.

Additionally WWF provides an out-of-box `SqlTrackingService` which is a simple track-and-store implementation, simply logging all described events to a database. This solution was considered as inappropriate, because of the vast number of events being triggered during a workflow execution. Simply logging all events and calculating the required values on demand, would result in a poor behavior - unacceptable for dynamic workflow composition.

WWF Tracking Service Functionality. The tracking service has to be attached to the workflow runtime. In standalone applications this can be accomplished by instantiating the service and adding it to the runtime in the source code. If the workflow is published as Web service (i.e. on a Microsoft

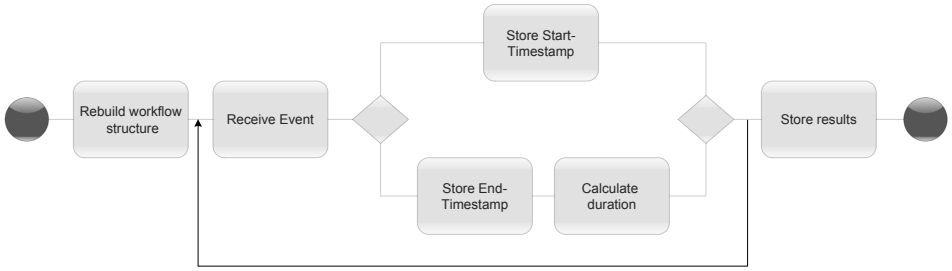

Figure 26: VRESCo Tracking Channel - Tracking Sequence

Internet Information Services (IIS) Server) the corresponding `Web.config` file has to be extended to attach the tracking service accordingly.
By initiating a workflow, the WWF tracking service instantiates a workflow tracking channel and passes all occurring events on to it. The channel represents the main component for implementing custom logging behavior.

4.3.3 VRESCo Tracking Service

The main functionality of the WWF workflow monitoring service is implemented in the tracking channel, which is used by tracking service.

Initialization. By initiating a new instance of a workflow its structure is mapped accordingly by building a tree-structure. The nodes of the tree represent *ActivityRecords*.
These records store the necessary information to significantly evaluate the Workflow. Each record is linked to the corresponding activity reference of the workflow, which facilitates the backtracking of workflow events.

On every occurrence of an event, the `ActivityExecutionStatus` is checked. If the status is `Executing`, the corresponding activity has been successfully initiated and has even now commenced its execution. In this case the VRESCo Tracking Channel stores the time stamp of this event in the corresponding `ActivityRecord` as the start time of the event. If the event's status is `Closed`, the activity terminated successfully and the VRESCO

```
1 using(WorkflowRuntime workflowRuntime = new WorkflowRuntime())
2 {
3
4     WFTrackingService tracking = new WFTrackingService(
          workflowRuntime);
5     workflowRuntime.AddService(tracking);
6     AutoResetEvent waitHandle = new AutoResetEvent(false);
7
8     workflowRuntime.WorkflowCompleted +=
9        delegate(object sender, WorkflowCompletedEventArgs e)
10       {
11          waitHandle.Set();
12       };
13    workflowRuntime.WorkflowTerminated +=
14       delegate(object sender, WorkflowTerminatedEventArgs e)
15    {
16       Console.WriteLine(e.Exception.Message);
17       waitHandle.Set();
18    };
19
20    WorkflowInstance instance =
21       workflowRuntime.CreateWorkflow(typeof(WPC_Casestudy.Workflow1)
             );
22
23    instance.Start();
24
25    waitHandle.WaitOne();
26 }
```

Listing 4: Workflow Monitoring - Example configuration for Self-Hosted Workflows

Tracking channel stores the terminating time stamp.

As for any typical tree structure, `ActivityRecords` also store their successors and predecessors. This tree allows to store the workflow structure and save it to the database. Other application, like the Workflow Monitoring Evaluation Tool (see Chapter 4.4) can use this information to recreate the workflow. Special types have been implemented for `IfElseActivities`, `ParallelActivities`, `SeuqenceActivites` and `WhileActivities`. According the different properties and behavior of certain Activities, also different types of `ActivityRecords` have been implemented to correctly map the underlying Workflow.. Additionally a `Hashtable` is used to index the `ActivityRecords`.

```
<WorkflowRuntime Name="WorkflowServiceContainer">
 <Services>
  <add type="
     System.Workflow.Runtime.Hosting.ManualWorkflowSchedulerService,
     System.Workflow.Runtime, Version=3.0.0.0,Culture=neutral,
     PublicKeyToken=31bf3856ad364e35" UseActiveTimers="true"/>

      ...

  <add type="VRESCo.Workflow.Tracking.WFTrackingService,
             VRESCo.Workflow, Version=1.0.0.0, Culture=neutral,
             PublicKeyToken=null"/>
 </Services>
</WorkflowRuntime>
```

Listing 5: Workflow Monitoring - Example Web.config configuration

Data Evaluation. The evaluation of QoS relevant data is kept simple. The evaluation of the *ExecutionTime* of an activity is accomplished as already explained above, by setting timestamps on certain activity execution status events respectively on *ActivityExecutionStatus.Executing* and *ActivityExecutionStatus.Closed*. The time span between these two timestamps represents the `ExecutionTime` of this Activity and is stored respectively in the database.
If the activity execution status is `Faulting` or `Canceling`, the execution time is not calculated. Instead the accuracy of this activity is decreased.

Tracking Store. The evaluated data is stored in a MS SQL database, using two tables. One table is used for workflow description and the other for data storage. Each invocation of an activity results in a database entry - if a workflow contains loops, there are several entries for activities enclosed by the loop.

4.3.4 Installation and Configuration

The configuration of workflows depends on their type of project. Workflows can be realized as host applications or be deployed as Web services on application servers (i.e. Microsoft IIS Server).

Self-Hosted Workflows. When developing host applications, the application itself handles the initialization of the workflow runtime. Thus, tracking services can be easily applied, since the direct reference to the runtime is at hand. Listing 4 shows how to setup the VRESCo Tracking Channel in a host application by passing the tracking service reference on to the workflow runtime.

Deploying Workflows as Web Services. When deploying workflows as Web services, the application server is responsible for initializing and maintaining the workflow runtime. The developed workflow has to provide a `Web.config` file, containing several instructions for the application server. To enable workflow tracking, the parameters of Listing 5 have to be applied to the configuration file.

4.4 Workflow Monitoring Evaluation

The previous chapter described how a workflow can be monitored and how its performance can be evaluated. This chapter will introduce two ways of evaluating the collected data.

4.4.1 Overview

The previous chapters introduced two different approaches to QoS monitoring. Chapter 4.1 presented a WPC-based QoS monitor, which takes advantage of the deeply into the Windows operating system integrated Windows Performance Counters. Due to this counters, the WPC-based QoS monitor provides highly accurate server side QoS data. Chapter 4.3 introduced a WWF workflow monitor, which makes use of a tracking service, which is provided by the Windows Workflow Foundation. This workflow monitor provides detailed client side performance data of all distinct activities of a workflow.

Both approaches evaluate and store their observations in databases. The WPC-based QoS monitor delivers its results to VRESCo where they are processed by the QoS Scheduler, which stores them in the registry database. The WWF workflow monitor stores its data in the tracking store. Querying a database to evaluate the results is very cumbersome, either way by using the command line or an integrated SQL-tool.

This chapter introduces a Windows application which allows for a continuous evaluation of a workflow and an evaluation API which can be used by other applications to query QoS information. The API, for instance, can be used by VRESCo to optimize its rebinding strategies. In combination with the `VRESCoWorkflowActivity`, introduced int Chapter 4.2, this enables VRESCo to provide highly effective dynamic Web service composition. This can be achieved, because all activities are taken into account. Furthermore, the composition is based on “life” data. Thus, VRESCo could react to changes of the QoS of certain activities, as well as rely on the number of iterations of loops.

4.4.2 Evaluation Tool

The evaluation tool provides a convenient way of visually evaluating a monitored workflow. The tool is a Windows desktop application with a graphical user interface, displaying all relevant data of the tracking store.

The evaluation tool consists of three parts:

- **Workflow Outline** - on the left side of the application is a tree panel (see Figure 27). This panel shows the exact representation of the workflow as tree. The workflow itself is the root node of the tree. Sequencing activities are displayed one below the other within the node of the enclosing `SequenceActivity`. `ParallelActivities` are nodes containing two `SequenceActivities`. Their representations are also stacked - according the WWF designer, which displays workflows vertically, represents the first node the left `SequenceActivity` of the `ParallelActivity` and the second node the right one. Each node of the tree shows the name of the activity.
- **The Activity Summary** - This panel is positioned on the top of the right side of the evaluation tool. It provides some basic statistics of the activities measured performance (minimum, maximum, range, mean, number of samples in database).
- **The Performance Histogram** - The performance histogram is located on bottom of the right side. It provides a graphical representation of the measured values. The range of a performance value (the distance between the minimum and maximum measured value) is divided into a predefined number of intervals. The graph shows the frequencies of values of each interval, thus providing a graphical representation of the distribution of the measured values.

The evaluation tool provides two dropdown fields which can be used to set the evaluation time frame. Initially the values of these fields are set to the timestamps of the first and last records of the corresponding workflow.

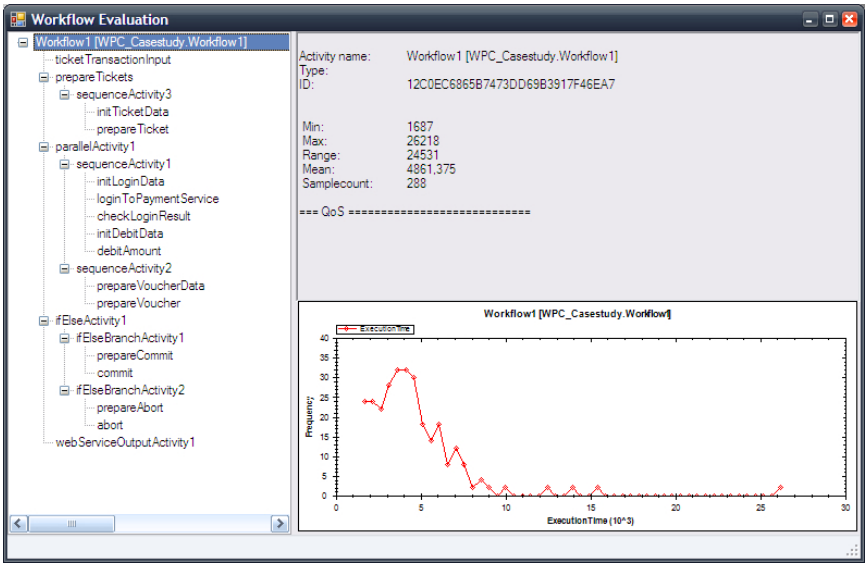

Figure 27: Workflow Evaluation Tool

For instance, these fields can be used to evaluate the performance of the workflow within the last 24 hours.

By clicking on an activity of the workflow outline, the activity summary on the right side of the application queries the tracking store for all entries of the specified time frame. Minimum, maximum and average values are calculated to basically describe the distribution of the observed data. Additionally a histogram is generated and displayed.
If the selected activity is of the type `WhileActivity`, additionally the average number of iterations of this loop is calculated and displayed. If the activity is a conditional `IfThenElseActivity`, the percentages of how often the then-path or the else-path has been chosen, is displayed. These values is required for QoS aggregation formulas (i.e. see Figure 30).

4.4.3 Evaluation API

The evaluation API provides performance data of workflow activities to other applications.

This API can be used to enhance the performance of VRESCo's dynamic workflow composition engine.

- **GetExecutionTime** - provides the average execution time of an activity within a specified time range (in milliseconds).
- **GetAccuracy** - provides the accuracy of an activity within a specified time range (in percent of successful executions).
- **GetThroughput** - provides the throughput of an activity within a specified time range (in executions).
- **GetWhileIterations.** - provides the average number of iteration of a `WhileActivity` within a specified time range (in number of iterations).
- **GetXORPercentage.** - provides the average percentage of how often the then-path of an IfThenElseActivity is executed, within a specified time range (in percent of then-path executions).

Method	Param	Description
GetExecutionTime	activityID	Identifier of the requested activity
	from	Start timestamp of the evaluation period
	to	End timestamp of the evaluation period
GetAccuracy	activityID	Identifier of the requested activity
	from	Start timestamp of the evaluation period
	to	End timestamp of the evaluation period
GetThroughput	activityID	Identifier of the requested activity
	from	Start timestamp of the evaluation period
	to	End timestamp of the evaluation period
GetWhileIterations	activityID	Identifier of the requested activity
	from	Start timestamp of the evaluation period
	to	End timestamp of the evaluation period
GetXORPercentage	activityID	Identifier of the requested activity
	from	Start timestamp of the evaluation period
	to	End timestamp of the evaluation period

Table 5: Evaluation API - Methods and Parameter

5 Evaluation

Chapter 4 listed and described the separate projects and described how they contribute to VRESCo or among each other. This chapter presents evaluations of the contributions of this thesis.

- *WPC monitoring versus hard coded measuring:* The performance of WPC monitoring is compared to hard coded execution time measurement, which is implemented directly in the Web services of the evaluation example (see Chapter 5.1). This comparison should show how precise WPC based monitoring is compared to direct measuring.
- *Workflow tracking versus WPC monitoring:* WPC monitoring is assumed to provide very accurate performance data of Web services. These values are compared with data evaluated through the WWF tracking service. This evaluation should point out if the Web service performance, measured on the providing host, is directly related to the performance of the Web service's invocation within a WWF workflow.
- *QoS aggregation algorithms versus workflow tracking:* As stated in Chapter 1.2 commonly used QoS aggregation algorithms take only activities into account, which invoke Web services. This evaluation step shows the discrepancy between predicted performance using standard algorithms and measured performance data.

To demonstrate the approaches, introduced by this thesis, a case study with an example implementation is used.

5.1 Case Study

The example used in the case study simulates the sales process of an online ticket store, like common Internet stores for cinema or concert tickets. The buying process is heavily simplified in order to provide a compact evaluation scenario. Normally such a process requires multiple user interactions and confirmations until any transaction can be initiated.

In this scenario it is assumed, that the customer has already confirmed his selection, agreed upon the general terms and conditions as well as supplied his credit card credentials. The Workflow as depicted in Figure 28 describes the remaining transaction which includes conducting an Internet payment provider, storing the acquisition information to the database and sending a voucher to the customer.

The business process relies on two Web services:

- **TicketService**. This service provides operations to handle actions related to tickets. They can be *prepared* for the transaction, a *voucher* can be prepared to be sent after the transaction has been *committed*. If there happens to occur an unexpected error, the transaction will be *aborted*.
- **PaymentService**. The payment of the tickets is executed by invoking an external accredited service provider. To properly execute a debit transaction, the user (TicketService) has to login successfully. The debit operation finally transfers the amount from the credit card owner's account to the account of the ticket service.
  This Web service is a perfect candidate for dynamically rebinding according to its performance, price, etc.

The sales process starts by recursively preparing each of the supplied tickets. After all tickets are prepared, the voucher is prepared to be sent to the customer. This is handled while logging in to the Payment service and waiting for result of the debit activity. According to the debit termination state, the whole transaction is either transmitted or aborted.

5.2 Example Implementation

The Case study consists of several separate projects:

- A *Setup Project*, hosting the previously exemplified Web services as well as the VRESCo Runtime. The Web services implement no real

functionality but simulate operation delays by waiting several milliseconds. The interval for operations that execute on single entities, consists of a predefined value.

- A *Workflow Project* composes the described sales process and exposes the Workflow as Web service. The service is hosted on an Microsoft IIS server. Auxiliary activities, which are necessary to prepare data for the Web service calls have also a simulated behavior. The workflow is designed using the newly introduced *VRESCoWebserviceActivity* (see Chapter 4.2).
- The workflow monitoring service (see Chapter 4.3) is used to measure the performance of the workflow.
- The WPC QoS monitoring service is used to measure the host-side performance of the Web service.
- A simple Java Web service client is implemented with Netbeans IDE[15]. This client recursively invokes the Workflow with random parameters.
- The Evaluation Tool (see Chapter 4.4) is used to evaluate the results of the testruns.

Runtime Behavior Simulation. Since the Web services don't implement real functionality, the runtime behavior of the activities is simulated. Sleep function calls are used to pretend a certain execution time. The interval is best guessed accordingly their functional description. `CodeActivities` are assumed to have a very short runtime because in this scenario they only prepare some data for the Web service calls.
It also depicts the complete simulated behavior a of the example workflow. According to this, the trivial assumption of one execution with only one ticket request is that the workflow will successfully execute in 1050ms.

[15] http://www.netbeans.org/

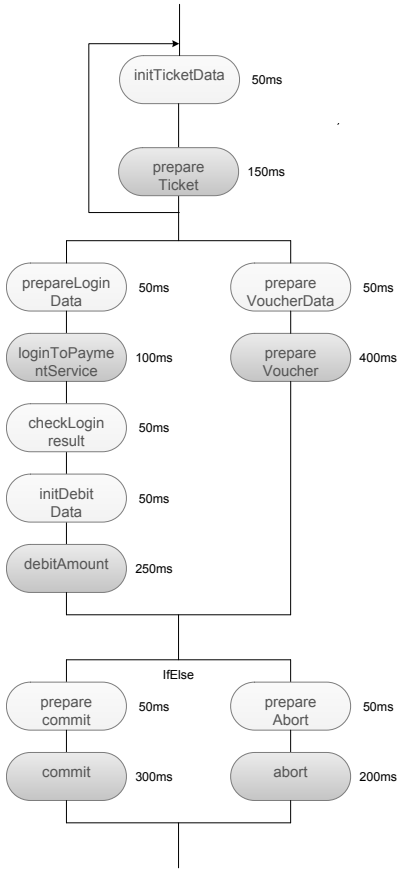

Figure 28: Evaluation Workflow

5.2.1 Evaluation Case Study Architecture

The complete structure of the evaluation case study - consisting of the three main parts Web service evaluation, workflow evaluation and overall evaluation - are implemented and set up according to the solution overview depicted in Figure 12.

The WPC case study setup project hosts the two Web services *TicketService* and *PaymentService* as well as a complete VRESCo runtime. On startup the VRESCo runtime and the two services are hosted and the services are

properly registered at VRESCo.

The example workflow is hosted on a Microsoft IIS Server with a properly configured workflow monitoring service. The evaluated data is stored in the tracking store.

A simple Java program invokes the workflow with alternating parameters. These parameters are controlling the behavior of the Web services. The first parameter supplies the number of ordered tickets, which corresponds to number of iterations of the first loop in the workflow. The second parameter represents the result of the payment operations.

5.2.2 Evaluation System

The Evaluation was performed on a Lenovo Thinkpad R61 laptop, with a Intel®Core™2 Duo CPU T8100 2.1GHz and 3GB main memory. This laptop was set up with Microsoft Windows XP SP3 and the .NET Framework 3.5. The VRESCo environment used in the evaluation has been built from source at Subversion revision number 17921. Microsoft SQL Server 9.0.4035 was used as data store for VRESCo as well as for the tracking store. Microsoft Internet-Information services (IIS) version 5.1 was used to host the evaluation workflow ad well as the WF tracking monitor. To evaluate and compare the data retrieved from the case study, Microsoft Excel 2007 was used.

5.3 Eval 1: WPC Monitoring versus Hard Coded Measuring

The first evaluation of this thesis examines the accuracy of WPC-based QoS monitoring. This approach to observing the performance of Web services is expected to provide precise results.

```
1 public bool CommitTransaction(Ticket ticket)
2 {
3     DateTime begin = DateTime.Now;
4
5     ...
6
7     DateTime end = DateTime.Now;
8     TimeSpan span = end - begin;
9
10    Console.WriteLine("code; TicketService; CommitTransaction; " +
          span.Milliseconds.ToString() + ";");
11
12    return true;
13 }
```

Listing 6: Eval 1 - Example of Hard Coded Performance Measurement

5.3.1 Evaluation Method

To evaluate if the QoS information observed by the WPC-based QoS monitor is valid and accurate, its results are compared to values retrieved from direct measuring.

Direct Measuring. Each method of the Web services implements its own time measuring and outputs its execution time to the console log (see Listing 6). At the beginning of each procedure, the current time stamp is stored. Just before the operation ends, or returns a result, another time stamp is taken. The difference of these two time stamps represents the execution time of the procedure. This information, as well as the Web service and procedure name, is formatted as Comma Separated Value (CSV) data and written to standard output.

WPC-based QoS Monitor Measuring. WPC-based QoS monitoring is used as explained in Chapter 4.1. An additional output statement is added to the `PublishPerformanceCounterQoS` method of the `QoSManagement-Service`, to track the results which are transmitted to VRESCo (see Listing 7). Again, this information is formatted as CSV-data and written to standard output. Furthermore, the *monitoringinterval* is set to 10 seconds to gather more data points for the evaluation.

```
1 public void PublishPerformanceCounterQoS(int revisionId, string
      opName, VReSCO.Contracts.UserObjects.PerformanceCounterQoS qos)
2 {
3    Console.WriteLine("wpc; " + revisionId + "; " + opName + "; " + (
         qos.ExecutionTime * 1000) + ";");
4    ...
```

Listing 7: Eval 1 - Extraction of WPC-based QoS Monitor Data

This evaluation uses the case study, as described above. The Java client invokes the example workflow 70 times with variable input parameters. Thus, the Web service operations `prepareVoucher`, `loginToPaymentService` and `debitAmount` are executed 70 times. Due to the variable input parameters of the workflow, `prepareTicket`, which is part of a `WhileActivity`, is executed 360 times. Similarly, the conditional `IfThenElseActivity` executed the `commit` operation 47 times and the `abort` operation 23 times.
Because the *monitoringinterval* has been set to 10 seconds, the WPC-based QoS monitor was invoked 31 times.

The standard output of the case study setup is redirected to a file which allows for a direct data import into Microsoft Excel.

5.3.2 Results

Direct measuring showed, that the execution times of the Web service methods are related to the predefined sleep-intervals of the implementation, though they vary about 5% which may result from the addidtional debugging output statements.

Table 6 lists the results of the two Web services. The first two columns display the names of the Web service's operations as well as their fixed delays (representing their simulated and expected execution time). The next column shows the execution times observed by direct measuring. The first value represents the minimum measured time and the second the maximum value. The third column shows the average execution time of all observed values retrieved through direct measuring (Avg(M)). The last column displays the mean value of all WPC-based QoS monitor results.

Activity	Fixed	Measured	Avg(M)	WPC
TicketService				
prepareTicket	150	156 - 171	160	160
prepareVoucher	400	421 - 437	428	429
commit	300	312 - 343	319	322
abort	200	203 - 218	214	214
PaymentService				
loginToPaymentService	100	101 - 109	106	107
debitAmount	250	265 - 281	268	268

Table 6: WPC Results

Comparing the min/max values in Table 6 and their average value with the result from WPC-based QoS monitoring, points out that a WPC performance value represents in three cases exactly the mean value of the execution times of the procedure. For the other three operations the WPC results are one to three milliseconds higher than these of direct measuring. One reason for this may be the time span between taking the second time stamp and the actual termination of the procedure (see Listing 6). This additional calculation as well as the I/O operation are not taken into account by direct measuring, whereas the according performance counter is updated by the actual termination of the procedure.

According to the collected data from direct measuring, the WPC value is nearly equal the average of the execution times within a *monitoringinterval*, with a deviation of less than one percent.

Thus, WPC performance monitoring as introduced in this thesis, represents a highly accurate tool for QoS measurement of WCF based Web services. Additionally, the possibility to apply this monitor to Web services without altering their implementations - moreover, the ability to apply it to binary implementations without access to the source code - makes it a valuable contribution to the VRESCo environment.

5.4 Eval 2: Workflow Tracking versus WPC Monitoring

Eval 1 showed that WPC-based QoS monitoring provides highly accurate performance measures. This step compares the performance of a Web service, which was observed with WPC-based QoS monitoring on the server side, with the performance of its invocation on the client side. This evaluation should give information about the performance of Web service invocations in WWF workflows.
The following points of interests have to be targeted:

1. What is the processing overhead of a Web service invocation?
2. Is this overhead constant for a single Web service?
3. Is this overhead constant for all Web services?

If the last question could be answered with "yes", it could be assumed, that the server side performance of a Web service is directly related to the performance on the client side. Thus, if the current QoS of the network is known, common QoS aggregation algorithms are expected to work efficiently.

5.4.1 Evaluation Method

To properly perform this evaluation, the Web services of the case study are observed by the WCF-based QoS monitor on the server side. The data is again extracted through additional CSV-formatted outputs (see *WPC-based QoS Monitor Measuring* in Chapter 5.3.1). This CSV-data is imported into Microsoft Excel and analyzed.
For measuring the performance of the Web service invocations within a WWF workflow, the WWF tracking monitor (see Chapter 4.3) is used.

Workflow Tracking. Workflow tracking is enabled according to Chapter 4.3. The evaluated data is stored in the tracking store as displayed in Figure

Activity	Fixed	WPC	Tracking	Diff.
TicketService				
prepareTicket	150	160	365	205
prepareVoucher	400	429	936	507
commit	300	322	522	200
abort	200	214	417	203
PaymentService				
loginToPaymentService	100	107	302	195
debitAmount	250	268	464	196

Table 7: WWF Tracking Results - Web Service Calls

12. The evaluation tool (see Chapter 4.4.2) is used to inspect the results of the tracking service.

Again the workflow is invoked 70 times by the Java client, with exactly the same input parameters as in Eval 1. This results in the same number of executions for the Web services as well as their operations. Accordingly, the same number of executions is observed on the client side by the WWF tracking monitor.

To evaluate the first question of the introduction the values are compared and the difference of the server side and client side execution time is calculated. To analyze if the calculated overhead is constant for a certain Web service, the evaluation tool is used, which already provides simple statistical analysis. Minimum and maximum values can be retrieved as well as a graphically distribution of the performance values. To answer question three, all invocation overheads are compared.

5.4.2 Results

According to question 1 the analysis of the observed performance data showed, that the overhead of invoking a Web service within a WWF workflow in this case study, spans from 195 milliseconds to 205 milliseconds and has a mean value of 200 milliseconds. Due to the fact, that Web services and workflow reside on the same host, this overhead results from querying

Activity	Range
TicketService	
prepareTicket	343 - 403
prepareVoucher	906 - 968
commit	500 - 562
abort	406 - 453
PaymentService	
loginToPaymentService	286 - 355
debitAmount	437 - 500

Table 8: Eval 2 - Ranges of Web Service Invocations

VRESCo and message processing – especially wrapping and unwrapping of DAIOs and SOAP messages. The value of `prepareVoucher` is disregarded by this calculation, as explained in the discussion (see Chapter 5.4.3).

Table 7 lists the results of the activities which call the example Web services *TicketService* and *PaymentService*. Again, the first column shows the hard coded delays of the Web services and the second the results from WPC-based QoS monitoring. The third column lists the results from the WWF tracking monitor. It represents the average execution time of all activity invocations.

To answer question 2, all minimum and maximum values of execution times of Web services are compared. This data has been observed by the WWF tracking monitor and is analyzed using the evaluation tool. Table 8 shows the ranges of all Web service operations. The mean range is 60 milliseconds. But, by analyzing the distributions of the observed performance values, by using the histogram visualization provided by the evaluation tool, it could be verified, that the values are largely located near the mean. An example distribution of this evaluation is depicted in Figure 29. It shows, that 45 times the value 453 was observed and 20 times the value 468. The mean value of this activity is 464, so 65 values (90%) of the values of the `debitAmount` activity of this case study are located near the mean.
According to this, it can be stated, that the invocation overhead is constant for a Web service.

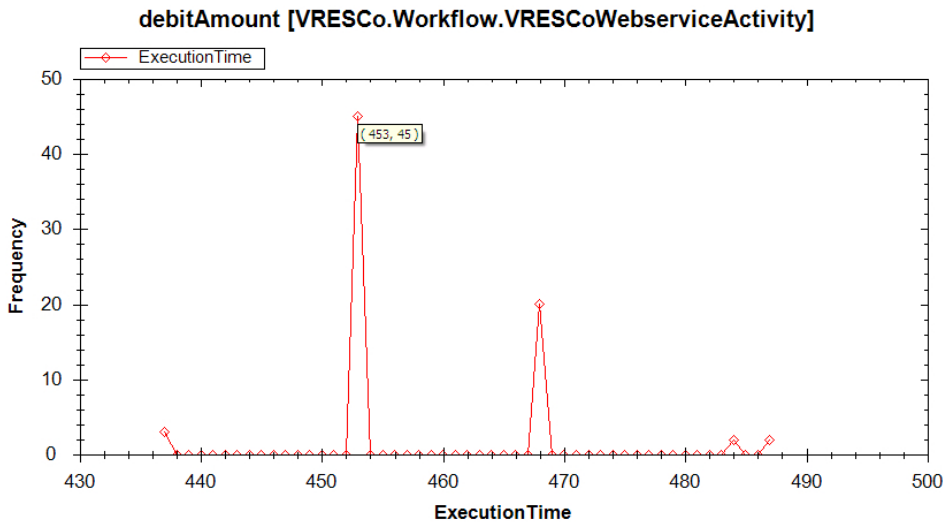

Figure 29: Eval 2 - Distribution of Execution Times of `DebitAmount`

By answering question one, it has already been shown, that the overhead of invoking a Web service spans from 195 milliseconds to 205 milliseconds and has a mean value of 200 milliseconds. With a standard deviation of 3,9 milliseconds, the overheads of all Web service invocations vary about 2% of the mean value. Thus, it can be assumed, that the overhead is constant for all Web services of this case study. Again `prepareVoucher` is disregarded by this calculation. The concerning activities

5.4.3 Discussion

All activities observed by the WWF tracking monitor performed equally according to their overhead of invoking Web services, except the first two activities of the `ParallelActivity` used in the case study. `PrepareVoucher` exceeded the average overhead by 250%. `PrepareVoucherData`, which does not invoke a Web service and just executes a few statements on the local host needed twice the time to execute as a similar activity before.
This behavior could be reproduced by simply moving the first sequence to the right side and the second to the left. In this scenario, `InitLoginData`

Attribute	Sequence	Loop	XOR-XOR	AND-AND
		Performance		
Response Time[a] (q_{rt})	$\sum_{i=1}^{n} q_{rt}(f_i)$	$q_{rt}(f) * c$	$\sum_{i=1}^{n} p_i * q_{rt}(f_i)$	$\max\{q_{rt}(f_1), .., q_{rt}(f_n)\}$
Throughput (q_{tp})	$\min\{q_{tp}(f_1), .., q_{tp}(f_n)\}$	$q_{tp}(f)$	$\sum_{i=1}^{n} p_i * q_{tp}(f_i)$	$\min\{q_{tp}(f_1), .., q_{tp}(f_n)\}$
Scalability (q_{sc})	$\min\{q_{sc}(f_1), .., q_{sc}(f_n)\}$	$q_{sc}(f)^c$	$\sum_{i=1}^{n} p_i * q_{sc}(f_i)$	$\min\{q_{sc}(f_1), .., q_{sc}(f_n)\}$

Figure 30: QoS Aggregation Formula

and `Login` activities perform almost equally.
An explanation of this behavior could not be found. Thus, these two values are disregarded in some calculations.

5.5 Eval 3: QoS Aggregation versus Workflow Tracking

The previous evaluation showed, that the performance of the Web service invoking activity on the client side is related to the performance on the server side. This evaluation will analyze the performance of the QoS aggregation formulas of Figure 30.

In the represented table, a sequence describes a `SequenceActivity` of the WWF - respectively a sequence of consecutive activities. The formula for response time q_{rt} states, that the response time of a sequence is equal the sum of all response times of the activities in this sequence $q_{rt}(f_i)$. For a loop - a `WhileActivity` in WWF - the response time q_{rt} is equal the product of the response time of the activity $q_{rt}(f)$ and the number of iterations c. For conditional Xor-Xor constructs - `IfThenElseActivities` in WWF - the response time is equal the sum of all response times of the activities $q_{rt}(f_i)$ multiplied with the probability of being invoked p_i. And-And construct - `ParallelActivities` in WWF - are calculated by selecting the maximum value of all contained activities.

5.5.1 Evaluation Method

To evaluate the performance of the aggregation algorithms of Figure 30, three approaches are analyzed. Each approach aggregates the QoS based on different input values.

Aggregation approach 1. The first approach takes the results of WPC-based QoS monitoring from Eval 1 and adds the additional overhead of 200 milliseconds from Eval 2 to each Web service invocation. For the first loop of the evaluation workflow it is assumed, that other observations indicated that the loop iterates four times on average. For the XOR-XOR activity of the evaluation workflow it is assumed, that the then-path is executed 60% of the time and the else-path respectively 40% of the time.

Aggregation approach 2. The second approach is equal to the first one, but it takes accurate input parameters for loops and XOR-XOR activities. By using the number of iterations, observed by the WWF tracking monitor, this approach should provide more accurate results than the first. Accordingly, the exact percentage of how often the then-path, respectively the else-path, is executed is provided for XOR-XOR activities. The exact number of iterations, calculated by the evaluation tool, is 5. The then-path of the `IfThenElseActivity` is executed 66,6% of the time and the else-path 33,3% of the time.

Aggregation approach 3. The third approach uses the exact input parameters from the second approach and includes the `CodeActivities` of the evaluation workflow. Therefore, the simplified assumption is made, that these activities have a general execution time of 50 milliseconds (based on the results of Table 10).

Acitivity	WPC	Aggr 1	Aggr 2	Aggr 3	Tracking
Loop	640	1440	1800	2150	2100
AND-AND	438	775	775	875	1550
Sequence 1	375	775	775	875	896
Sequence 2	438	638	638	738	1056
XOR-XOR	260	460	466	516	540
Sequence 3	300	500	500	550	575
Sequence 4	200	400	400	450	465
Workflow	1338	2675	3041	3541	4190

Table 9: Eval 3 - Comparing Aggregation Algorithms

5.5.2 Results

Table 9 shows the results of the aggregation approaches according to their input parameters.
The first line shows the results for the loop/`WhileActivity` of the evaluation workflow. The first value represents the value observed by the WPC-based QoS monitor multiplied with the assumed number of iterations. The second value is calculate according aggregation approach one and takes the processing overhead into account. The third value is based on the number of iterations as observed by the WWF tracking monitor. The fourth line is based on aggregation approach three and takes also `CodeActivities` into account. And the last line shows the execution time measured by the WWF tracking monitor.
As already can be seen, the more precise the input parameters, the more accurate the results of the aggregation. Even aggregation approach two, which does not take auxiliary activities into account, provides a very accurate estimate of the execution time of this loop.
The results of the AND-AND/`ParallelActivity` - the second line in Table 9 - are more complex. The result of the AND-AND aggregation is the maximum of *Sequence 1* and *Sequence 2*. If only QoS data of the WPC-based QoS monitor is used, *Sequence 2* would take longer to execute. Adding the processing overhead to each Web service invocation will pick *Sequence 1* as maximum. Approach two does not provide more accurate input, thus the result is the same. Aggregation approach three provides again highly accurate performance estimations, except for *Sequence 2* (refer to the discussion

Activity	Fixed	Tracking
initTicketData	50	50
initLoginData	50	50
checkLoginResult	50	51
prepareVoucherData	50	101
prepareCommit	50	50
prepareAbort	50	50

Table 10: WWF Tracking Results - Auxiliary Activities

in Chapter 5.4.3). According to these results a workflow optimization based on WPC-based QoS data would choose *Sequence 2* for further optimization. Based on further information from the WWF tracking monitor, *Sequence 1* would be selected.
The XOR-XOR/`IfThenElseActivity` showed now noticeable abnormalities, but only improved the accuracy of the aggregation algorithms by providing more accurate input parameters.
The last line represents the sum of the loop, AND-AND and XOR-XOR activities. This also reflects the assumption: the more accurate the input parameters, the more accurate the result of the aggregation algorithm.

This evaluation showed that performance data observed by the WWF tracking monitor provides valuable input for aggregation algorithms used by VRESCo. This data could be queried by VRESCo, by using the evaluation API introduced in Chapter 4.4.3.

6 Conclusion and Future Work

Selecting component services of a set of semantical equal Web services to optimize the overall QoS of a composite service, commonly representing a value contributing business process, is one of the main objectives of QoS-aware service compositions. In order to meet this expectations QoS aggregation algorithms need to based on profound data which is gathered from measuring the performance of the corresponding component services. A vast amount of literature describes various methods of QoS aggregation and QoS data repositories but only a few approaches focus on the process of performance measuring.

This thesis introduced two new approaches of measuring the QoS of WCF based Web services. One solution, the WPC monitoring service, makes use of Windows Performance Counters, which are integrated into the operating system and provide highly accurate performance measures at a minimum of additional computational overhead. The evaluation proved that performance values retrieved from the WPC monitor are as accurate as hard coded performance measuring directly integrated into the application.
The second approach makes advantage of the tracking service, provided by the WWF, which tracks all events, that are initiated during the execution of a workflow. Contrary to WPC monitoring, which evaluates the performance of component services directly on the hosting server, the WF tracking server resides in the application server that hosts the workflow, thus representing a client side view of the component services. The evaluation showed, that the performance of Web service invocations is related to the performance measured by the WPC monitor, taking a processing and communication overhead into account. It also pointed out, how traditional QoS aggregation algorithm estimates deviate sometimes more then 100% for not taking auxilliary activities into account.

6.1 Future Work

The evaluation showed how estimated QoS values can deviate from measured performance. Based on the data observed by the WF tracking service, new

algorithms have to be evaluated which take auxiliary activities into account and take advantage of the Evaluation API provided by this thesis.

Appendix

A List of Abbreviations

AOP	Aspect Oriented Programming
API	Application Programming Interface
BPEL	Business Process Execution Language
BPI	Business Process Integration
BPM	Business Process Management
BPML	Business Process Modeling Language
BPMS	Business Process Management Systems
Caas	Composition as a Service
CPU	Central Processing Unit
Daios	Dynamic and asynchronous invocation of services
ebXML	Electronic Business using eXtensible Markup Language
EMSC	Extended Message Sequence Charts
ER	Enterprise Relationship
FTP	File Transfer Protocol
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IIS	Microsoft Internet Information Services
ISO	International Organization for Standardization
IT	Information Technology
JAX-WS	Java API for XML based Web Services
JAXB	Java Architecture for XML Binding
JMX	Java Management Extension
MSC	Message Sequence Charts
OASIS	Organization for the Advance of Structured Information Standards
OWL	Web Ontology Language
P2P	Peer-to-Peer
QoS	Quality of Service
RDF	Resource Description Framework
REST	Representational state transfer

RPC	Remote Procedure Call
SLA	Service Layer Agreement
SMTP	Simple Mail Transfer Protocol
SOA	Service oriented Architecture
SOC	Service Oriented Computing
SQL	Structured Query Language
TCP	Transmission Control Protocol
UBR	UDDI Business Registry
UDDI	Universal Description, Discovery and Integration
URL	Uniform Resource Locator
VCL	Vienna Composition Language
VQL	VRESCo Query Language
VRESCo	Vienna Runtime Environment for Service oriented Computing
W3C	World Wide Web Consortium
WCF	Windows Communication Foundation
WPC	Windows Performance Counter
WS-BPEL	Web Services Business Process Execution Language
WS-CDL	Web Services Choreography Description Language
WSDL	Web Services Description Language
WWF	Windows Workflow Foundation
WWW	World Wide Web
XML	eXtensible Markup Language

B WCF Performance Counters

```
1 Transacted Operations Committed
2 Transacted Operations Aborted Per Second
3 Calls Failed
4 Queued Messages Rejected Per Second
5 Transacted Operations In Doubt Per Second
6 Transacted Operations In Doubt
7 Security Calls Not Authorized
8 Transacted Operations Committed Per Second
9 Security Validation and Authentication Failures
10 Calls Duration
11 Transacted Operations Aborted
12 Queued Messages Dropped
13 Reliable Messaging Sessions Faulted Per Second
14 Queued Poison Messages Per Second
15 Calls Faulted Per Second
16 Instances
17 Security Calls Not Authorized Per Second
18 Instances Created Per Second
19 Calls Outstanding
20 Calls Faulted
21 Calls
22 Calls Failed Per Second
23 Reliable Messaging Sessions Faulted
24 Transactions Flowed Per Second
25 Queued Messages Rejected
26 Security Validation and Authentication Failures Per Second
27 Queued Poison Messages
28 Reliable Messaging Messages Dropped Per Second
29 Calls Per Second
30 Queued Messages Dropped Per Second
31 Reliable Messaging Messages Dropped
32 Transactions Flowed
```

Listing 8: ServiceModelService 3.0.0.0

```
1 Calls
2 Calls Per Second
3 Calls Outstanding
4 Calls Failed
5 Calls Failed Per Second
6 Calls Faulted
7 Calls Faulted Per Second
8 Calls Duration
9 Reliable Messaging Messages Dropped
10 Reliable Messaging Messages Dropped Per Second
11 Reliable Messaging Sessions Faulted
12 Reliable Messaging Sessions Faulted Per Second
13 Security Calls Not Authorized
14 Security Calls Not Authorized Per Second
15 Security Validation and Authentication Failures
16 Security Validation and Authentication Failures Per Second
17 Transactions Flowed
18 Transactions Flowed Per Second
```

Listing 9: ServiceModelEndpoint 3.0.0.0

```
1 Calls
2 Calls Per Second
3 Calls Outstanding
4 Calls Failed
5 Calls Failed Per Second
6 Calls Faulted
7 Calls Faulted Per Second
8 Call Duration
9 Security Validation and Authentication Failures
10 Security Validation and Authentication Failures Per Second
11 Security Calls Not Authorized
12 Security Calls Not Authorized Per Second
13 Transactions Flowed
14 Transactions Flowed Per Second
```

Listing 10: ServiceModelOperation 3.0.0.0

C VRESCo Client Library - Example Invocation

```
1
2  IVReSCOQuerier querier =
3     VReSCOClientFactory.CreateQuerier("guest");
4
5  IVReSCOPublisher publisher =
6     VReSCOClientFactory.CreatePublisher("guest");
7
8  // building the request message
9  DaiosMessage smsRequest = new DaiosMessage();
10 smsRequest.SetString("RecipientNumber", "0699-12341242");
11 smsRequest.SetString("SenderNumber", "0650-69696669");
12 smsRequest.SetString("Message", delay);
13
14 DaiosMessage SendSMS = new DaiosMessage();
15 SendSMS.SetComplex("SMSRequest", smsRequest);
16
17
18 // create rebinding-proxy
19 var fquery = new VQuery(typeof(Feature));
20 fquery.Add(Expression.Eq("Name", "SendSMS"));
21 IList<Feature> features =
22    querier.FindByQuery(fquery, QueryMode.Exact) as IList<Feature>;
23
24
25 IList<ServiceRevision> revisions = querier.GetAllRevisions();
26 foreach (ServiceRevision rev in revisions)
27 {
28    publisher.DeactivateRevision(rev.Id);
29 }
30
31
32 var frquery = new VQuery(typeof(ServiceRevision));
33 frquery.Add(Expression.Eq("Operations.Feature.Name", "SendSMS"));
34
35
36 DaiosRebindingMappingProxy proxy =
37    querier.CreateRebindingMappingProxy(frquery,
38                                        QueryMode.Exact,
39                                        0,
40                                        new
                                              MappingRebindingStrategy
                                              ()
41    ) as DaiosRebindingMappingProxy;
42
43
44 proxy.UseMapping = true;
45
46
47 foreach (ServiceRevision rev in revisions)
48 {
49     if (rev.Service.Name == name)
50     {
51         publisher.ActivateRevision(rev.Id);
52     }
53     else
54     {
55         publisher.DeactivateRevision(rev.Id);
56     }
```

```
57 }
58
59 proxy.ForceRebinding();
60
61
62 // invoke service
63 DaiosMessage result = proxy.RequestResponse(SendSMS);
```

Listing 11: Example - VRESCo Web service Invocation

References

[1] Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.

[2] Natee Artaiam and Twittie Senivongse. Enhancing service-side qos monitoring for web services. In *SNPD '08: Proceedings of the 2008 Ninth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*, pages 765–770, Washington, DC, USA, 2008. IEEE Computer Society.

[3] Luciano Baresi, Carlo Ghezzi, and Sam Guinea. Smart monitors for composed services. In *ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing*, pages 193–202, New York, NY, USA, 2004. ACM.

[4] Bruce Bukovics. *Pro WF: Windows Workflow in .NET 3.5.* Apress, Berkely, CA, USA, 2008.

[5] Michael J. Carey. Soa what? *Computer*, 41(3):92–94, 2008.

[6] Issam Chebbi, Schahram Dustdar, and Samir Tata. The view-based approach to dynamic inter-organizational workflow cooperation. *Data Knowl. Eng.*, 56(2):139–173, 2006.

[7] OASIS International Standards Consortium. Uddi version 3.0.2, September 2004. http://www.uddi.org/pubs/uddi_v3.htm.

[8] OASIS International Standards Consortium. ebxml registry services and protocols v3.0, March 2005. http://www.ebxml.org.

[9] Thomas Erl. *Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.

[10] Li Fei, Yang Fangchun, Shuang Kai, and Su Sen. A policy-driven distributed framework for monitoring quality of web services. In *ICWS '08: Proceedings of the 2008 IEEE International Conference on Web Services*, pages 708–715, Washington, DC, USA, 2008. IEEE Computer Society.

[11] Organization for the Advance of Structured Information Standards (OASIS). Reference model for service oriented architecture 1.0, oasis standard, October 2006. http://docs.oasis-open.org/soa-rm/v1.0/.

[12] Apache Software Foundation. Apache axis, 07 2009. http://ws.apache.org/axis/.

[13] David Gristwood. Windows workflow foundation: Tracking services introduction, January 2007. http://msdn.microsoft.com/en-us/library/bb264459

[14] Lican Huang, D.W. Walker, O.F. Rana, and Yan Huang. Dynamic-workflow management using performance data. In *Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on*, volume 1, pages 154–157, May 2006.

[15] Nicolai Josuttis. *Soa in Practice: The Art of Distributed System Design.* O'Reilly Media, Inc., 2007.

[16] Philipp Leitner, Florian Rosenberg, and Schahram Dustdar. Daios: Efficient dynamic web service invocation. *IEEE Internet Computing*, 13(3):72–80, 2009.

[17] Phillip Leitner. The daios framework - dynamic, asynchronous and message-oriented invocation of web services. Master's thesis, Vienna University of Technology, 10 2007.

[18] Fei Li, Fangchun Yang, Kai Shuang, and Sen Su. Q-peer: A decentralized qos registry architecture for web services. In *ICSOC '07: Proceedings of the 5th international conference on Service-Oriented Computing*, pages 145–156, Berlin, Heidelberg, 2007. Springer-Verlag.

[19] MSDN Library. Performance counters in the .net framework. Webpage, July 2009. http://msdn.microsoft.com/en-us/library/w8f5kw2e.aspx.

[20] Anton Michlmayer, Florian Rosenberg, Phillip Leitner, and Schahram Dustdar. End-to-end support for qos-aware service selection, invocation and mediation in vresco. 05 2009.

[21] Anton Michlmayr, Philipp Leitner, Florian Rosenberg, and Schahram Dustdar. Publish/subscribe in the vresco soa runtime. In *DEBS '08: Proceedings of the second international conference on Distributed event-based systems*, pages 317–320, New York, NY, USA, 2008. ACM.

[22] Anton Michlmayr, Florian Rosenberg, Philipp Leitner, and Schahram Dustdar. Qos-aware service provenance inweb service runtimes. 2009.

[23] Anton Michlmayr, Florian Rosenberg, Christian Platzer, Martin Treiber, and Schahram Dustdar. Towards recovering the broken soa triangle: a software engineering perspective. In *IW-SOSWE '07: 2nd international workshop on Service oriented software engineering*, pages 22–28, New York, NY, USA, 2007. ACM.

[24] Microsoft.com. Uddi shutdown faq, July 2009. http://uddi.microsoft.com/about/FAQshutdown.htm.

[25] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-oriented computing: State of the art and research challenges. *Computer*, 40(11):38–45, November 2007.

[26] Chris Peiris, Dennis Mulder, Amit Bahree, Aftab Chopra, Shawn Cicoria, and Nishith Pathak. *Pro WCF: Practical Microsoft SOA Implementation (Pro)*. Apress, Berkely, CA, USA, 2007.

[27] Chris Peltz. Web services orchestration and choreography. *Computer*, 36(10):46–52, 2003.

[28] Franco Raimondi, James Skene, and Wolfgang Emmerich. Efficient online monitoring of web-service slas. In *SIGSOFT '08/FSE-16: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pages 170–180, New York, NY, USA, 2008. ACM.

[29] World Wide Web Consortium (W3C) recomendation. Web services description language (wsdl) 1.1, March 2001. http://www.w3.org/TR/wsdl.

[30] World Wide Web Consortium (W3C) recomendation. Web services choreography description language version 1.0, November 2005. http://www.w3.org/TR/ws-cdl-10/.

[31] World Wide Web Consortium (W3C) recomendation. Soap version 1.2 part 0: Primer (second edition), April 2007. http://www.w3.org/TR/2007/REC-soap12-part0-20070427/.

[32] Red Hat, Inc. *Hibernate Reference Documentation v3.3.1*, 2008.

[33] Michael Rosen, Boris Lublinsky, Kevin T. Smith, and Marc J. Balcer. *Applied SOA: Service-Oriented Architecture and Design Strategies*. Wiley Publishing, 2008.

[34] F. Rosenberg, C. Platzer, and S. Dustdar. Bootstrapping performance and dependability attributes ofweb services. In *Proc. International Conference on Web Services ICWS '06*, pages 205–212, September 18–22, 2006.

[35] Florian Rosenberg, Philipp Leitner, Anton Michlmayr, Predrag Celikovic, and Schahram Dustdar. Towards composition as a service - a quality of service driven approach. In *ICDE '09: Proceedings of the 2009 IEEE International Conference on Data Engineering*, pages 1733–1740, Washington, DC, USA, 2009. IEEE Computer Society.

[36] W3C Member Submission. Owl-s: Semantic markup for web services, November 2004. http://www.w3.org/Submission/OWL-S/.

[37] Mingjie Sun, Bixin Li, and Pengcheng Zhang. Monitoring bpel-based web service composition using aop. In *ICIS '09: Proceedings of the 2009 Eigth IEEE/ACIS International Conference on Computer and Information Science*, pages 1172–1177, Washington, DC, USA, 2009. IEEE Computer Society.

[38] Liangzhao Zeng, B. Benatallah, A.H.H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. Qos-aware middleware for web services composition. *Software Engineering, IEEE Transactions on*, 30(5):311–327, May 2004.

[39] Liangzhao Zeng, Hui Lei, and Henry Chang. Monitoring the qos for web services. In *ICSOC '07: Proceedings of the 5th international conference on Service-Oriented Computing*, pages 132–144, Berlin, Heidelberg, 2007. Springer-Verlag.

[40] Wentao Zhang, Yan Yang, Shengqun Tang, and Lina Fang. Qos-driven service selection optimization model and algorithms for composite web services. In *COMPSAC '07: Proceedings of the 31st Annual International Computer Software and Applications Conference*, pages 425–431, Washington, DC, USA, 2007. IEEE Computer Society.

[41] Farhana Zulkernine and Patrick Martin. Conceptual framework for a comprehensive service management middleware. In *AINAW '07: Proceedings of the 21st International Conference on Advanced Information Networking and Applications Workshops*, pages 995–1000, Washington, DC, USA, 2007. IEEE Computer Society.

[42] Farhana H. Zulkernine, Patrick Martin, and Kirk Wilson. A middleware solution to monitoring composite web services-based processes. In *SERVICES-2 '08: Proceedings of the 2008 IEEE Congress on Services Part II*, pages 149–156, Washington, DC, USA, 2008. IEEE Computer Society.