FAKULTÄT FÜR !NFORMATIK

A Domain-Specific Language for QoS-Aware Service Composition

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Information Computing

eingereicht von

Predrag Celikovic
Matrikelnummer 0227192

an der
Fakultät für Informatik der Technischen Universität Wien

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

Wien, 12.01.2010

(Unterschrift Verfasser) (Unterschrift Betreuer)

Declaration

Predrag Celikovic
Linzer Straße 46
1140 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift)

Abstract

Software as a Service is a software deployment and distribution model that has grown in the last few years. It has become a valid business model to expose and license a specialized Web service on demand to customers. Companies can build software systems out of internal services or software components and use external services offered by third-party companies. Composition of these services is an important topic because of the rapid increase of involved Web services in enterprise software systems. Composition languages such as BPEL and others have been developed in the last few years.

This masters thesis presents a CaaS (Composition as a Service) approach for Web service compositions. The VCL (Vienna Composition Language) domain-specific language is introduced. The main focus of VCL and this thesis lies on microflows. Microflows are simple service compositions without support of user interaction at execution time. The VCL is used to specify microflows.

A composition services takes the VCL composition and transforms it to a Microsoft Windows Workflow which will be hosted as a own Web service in the IIS (Internet Information Server). In VCL, dependencies between services are defined implicitly based on the input and output parameter of those. However, by adding control structures it is possible to override the dataflow dependencies and modify the resulting composition.

To fulfill the SOA "Publish-Find-Bind" principal, VRESCo is used as a service registry. in addition, to a classical service registry like UDDI, VRESCo supports an abstraction layer for Web services that is used to map different service implementation from the same domain to a common interface definition. VCL specifies the composition on this abstract layer and allows to add, unlike other composition languages, QoS constraints. Therefore, a constraint hierarchy system with optional and mandatory constraints support is implemented to avoid over-constraint compositions. The composition service takes the given VCL composition description, resolve the functional requirements (e.g., input, output requirements) and choose the best solution based on the given QoS constraints.

Zusammenfassung

Software as a Service ist ein Software-Distributions-Modell welches in den letzten Jahren an Bedeutung gewonnen hat. Es wurde zu einem erfolgreichen Geschäftsmodell indem man Software als Dienstleistung in Form von standardisierten Web Services anbietet. Unternehmen können dadurch in komplexen Softwaresystem für bestimmte Aufgaben auf Web Services von externen, spezialisierten Unternehmen zurückgreifen. Durch die immer größer werdende Anzahl an verwendeten Web Services in großen Unternehmensanwendungen wurde auch die Komposition von diesen zu einem wichtigen Thema. Um dies zu ermöglichen entstanden spezielle Kompositionssprachen.

Diese Diplomarbeit stellt ein CaaS (Composition as a Service) Ansatz für Web Service Kompositionen vor. Um Kompositionen zu definieren wurde die Sprache VCL (Vienna Composition Language) entwickelt. Der Hauptaugenmerk der Arbeit liegt auf einfachen Microflows. Ein Microflow ist eine einfache, atomare Web Service Komposition die es den Benutzer, anders als BPEL, nicht ermöglicht die Komposition zur Ausführungszeit noch zu beeinflussen.

Ein Kompositionsservice transformiert die spezifizierte VCL Komposition zu einem Microsoft Windows Workflow und stellt ihn auf dem IIS (Internet Information Server) als ein eigenständiges Web Service zur Verfügung. In VCL werden Abhängigkeiten zwischen Web Services basierend auf den Eingabe- und Ausgabeparameter implizit definiert. Jedoch ist es möglich durch die Verwendung von Kontrollstrukturen die resultierende Komposition zu verändern.

Um das “Publish-Find-Bind” SOA Prinzip zu erfüllen wird VRESCo als ein Service Register verwendet. Anders als UDDI, abstrahiert VRESCo verschiedene Service Implementierungen derselben Domäne auf eine gemeinsame Darstellung. VCL unterstützt, anders als andere Kompositionssprachen, auch die Angabe von gewünschten QoS Werten. Hierfür wird ein so genanntes Constraint Hierarchy System verwendet das es dem Benutzer ermöglichen Kriterien zu spezifizieren die erforderlich oder nur optional sind. Das Composition Service evaluiert eine in VCL angegebene Komposition nach funktionalen Anforderungen (Eingabe- und Ausgabeparameter) und wählt anhand der QoS Kriterien die beste Komposition aus.

Danksagung

Ich möchte diese Diplomarbeit meinen Eltern Marica und Stojan widmen, die mich während des ganzen Studiums in jeder Hinsicht unterstützt haben. Sie haben mir das Studium ermöglicht und alle meine Entscheidungen unterstützt und mitgetragen.
Bedanken möchte ich mich bei meiner Verlobten Gordana die soviel Verständnis für meine Arbeit aufbrachte und oft auf mich verzichten musste.
Ein ganz besonderer Dank gilt meinem Betreuer Florian Rosenberg für die hervorragende Betreuung und Zusammenarbeit. Es hat Spaß gemacht an den verschiedenen Konzepten und Implementierungen zu arbeiten. Die Betreuung während der ganzen Diplomarbeit war mehr als hervorragend.
Desweiteren möchte ich mich bei meinem Professor Schahram Dustdar bedanken der es ermöglichte diese Arbeit am Institut für Informationssysteme über einen längeren Zeitraum und neben der Berufstätigkeit zu machen.

Ovaj diplomski rad posvećujem svojim roditeljima Marici i Stojanu, koji su mi omogućili studiranje i podržavali me u svakom pogledu tokom cijelog mog školovanja.
Zahvaliti želim i svojoj zaručnici Gordani, koja je imala puno razumijevanja i strpljenja za mene i moj rad.
Posebno želim zahvaliti svome mentoru Florianu Rosenbergu na izvanrednom savjetovanju i saradnji. Bilo je interesantno raditi na različitim konceptima i implementacijama. Podrška tokom izrade ovog rada bila je više nego odlična.
Želim još da zahvalim profesoru Schahramu Dustdaru, koji je omogućio izradu ovog diplomskog rada na Institutu za informacione sisteme.

Contents

List of Figures

Listings

1 Introduction

In the past 30 years we have been faced with a rapid development in computer technology. Computer networks grow with incredible speed. In the mid nineties the Internet began to influence our daily life. Today, the Internet is a common communication media that is used for many different activities. To buy a book it is no longer necessary to go to the library. Online Shops like Amazon allow the buyer to order books and other goods easily from home and available 24 hours per day, 365 days per year. E-Governance portals provide us with the possibility to file a tax return or change our living address online.

To make such services possible, highly sophisticated distributed software systems are necessary. Information and resources are needed to be shared between many computers. Starting with simple file and printer sharing, the goal to save resources was achieved. But in typical complex software solution many computers, software parts and sources are involved. It was necessary to establish a way of communication between those different parts.

In today's complex economy, business models require huge information exchange. In the whole chain of building a car there are often more than 100 different suppliers involved, leading to big enterprise software solutions. The software architecture moved from classic standalone software to distributed software systems. Different standalone software parts now require to communicate with each other. Moving away from simple file exchange over a network the idea of RPC (Remote Procedure Call) [72] was born. RPC is used for an inter-process communication between two different computer programs. It allows calling a procedure on another computer over a connected network. For many programming languages a RPC Implementation like CORBA (Common Object Request Broker Architecture) [30], Microsoft .NET Remoting [58], RMI (Java Remote Method Invocation) [28] or ONC RPC [64] is available.

One of the drawbacks of RPC is the architecture of the software that is using it. It looked like the old standalone solution with the attempt to hide the network between two computers and processes. A huge enterprise solutions wasn't maintainable with RPC. The idea of SOA (Service-oriented architecture) appeared. It introduces the word "Service", which stands for an independent, interoperable, loosely coupled software unit. A service is offering a defined functionality over a standardize protocol. It has strict boundaries, which simplifies coding, maintaining and deployment responsibilities.

A service consumer can bind to a service and consume it. Inspired by object-oriented programming the service is offered as an interface to the consumer. The clear advantage is the hiding of the implementation details. The service can be developed

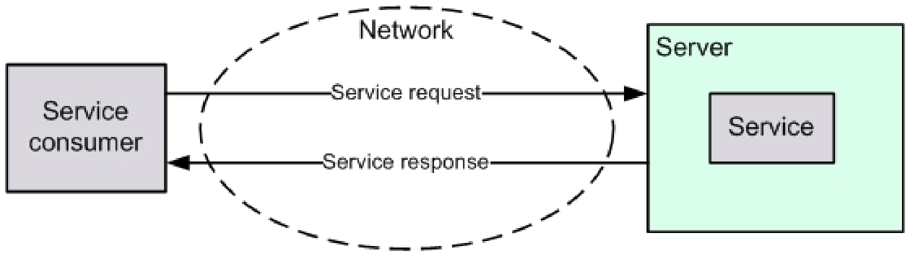


Figure 1: Service Call

by a second development team within the organization or generally provided by a third-party company.

To increase the interoperability between different operating systems, programming language and processor architectures a standardized way of communication is defined. The service is offering metadata information about the exposed interface, in the majority of cases it is WSDL (Web Service Description Language). Today, WSDL is “de facto” academic and industry standard. WSDL is a XML-based language that is used to describe the service interfaces with all details that are necessary to bind to a service. For the communication itself SOAP (Simple Object Access Protocol) is used. It is also XML-based and it is responsible for a correct exchange of the transported data. SOAP is mostly used in conjunction with HTTP (Hypertext Transfer Protocol) as transport layer.

Based on these standards it is possible to consume a service implemented in .NET and hosted on a Windows operating system within the IIS server from a Java application. This essentially simplifies the communication between different companies and completely new business models emerge. Third-party companies are now offering services that can be integrated in individual applications without big effort. The cost of in-house developing, testing and maintenance disappear. The requirements of experts in a specialized area can be easily outsourced.

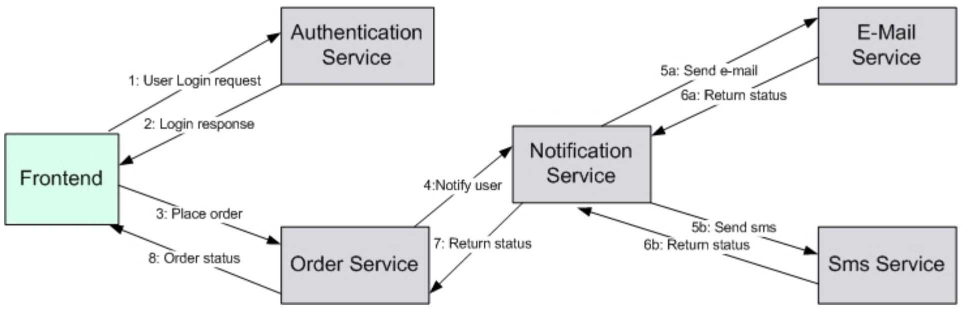


Figure 2: Service Composition

By combining services with each other a whole system can be created. Figure 2 show a small application example that is using services. Every service can be maintained by different employees, teams or even companies. The frontend developer only needs to know about the authentication and order service. The SMS service can also be a service provided by a third-party company. Internal changes of a service, like bug fixing or changing of business logic, will not affect the application. If a new frontend is required, a mobile client for example, all backend services can be reused. Only a new frontend needs to be developed. This increases the whole development process.

Service-oriented Architecture simplifies today's software development and raises it to a higher level. But SOA also has it difficulties, managing such complex systems is not a trivial task. As soon as a network between two or more parts (e.g., web services) is involved, topics like reliability, availability, security, bandwidth or scalability get more important. Web services can be hosted on different environments. A Web service can be load-balanced in a server farm on many servers. This increases the availability. Different authentication methods like classical Username-Password or Token Based Authentication can be supported by web services. All these characteristics are commonly grouped as Quality of Service characteristics. Having many services in a distributed system the right way of orchestration by considering QoS is crucial.

1.1 Motivation

SOA has a big influence on today's software development. But not all principles are widely accepted. One of SOAs fundamental principles is "Publish-Find-Bind". The idea behind it is to support loosely coupled services. In Figure 1 a hard-wired service is shown. The service consumer knows about the endpoint and the interface of the service. Moreover the service endpoint address is hard-coded on the service side. This leads to maintainability problems when a service endpoint changes. If a service consumer is for example a Windows client application consuming a service that is hosted on a public accessible server and the client is used by many users, an endpoint change will lead to broken clients.

A solution for that is to use a service registry that is acting as a broker. An example is the standardized UDDI (Universal Description, Discovery and Integration) registry. Figure 3 depicts such an infrastructure. A service provider registers a service at the service registry. A consumer contacts the service registry with an appropriate query for a service. The services registry provides, if a match is found, all necessary service information that is required to use the service to the consumer. The above mentioned Windows client will benefit from a service registry. The used service is decoupled, and can change his location. The only part that needs to be notified is the service registry. For the Windows client a location change will be completely transparent.

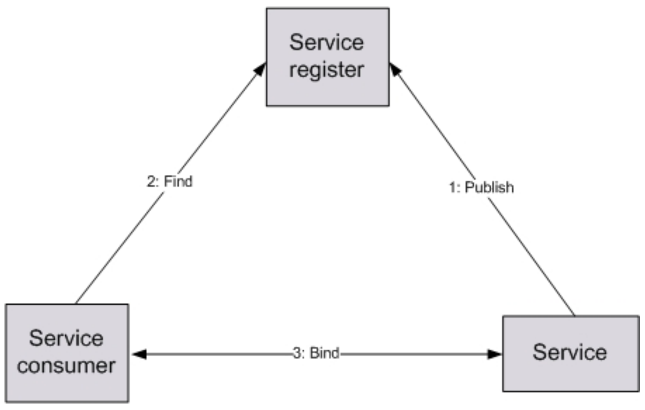


Figure 3: Publish Find Bind

Nevertheless, UDDI never reached high acceptance by the industry. Microsoft, SAP and IBM closed their public UDDI service registers in January 2006. A flat service structure without support for versioning, an expressive query language and the absence of support for non-functional characteristics of hosted services are some of main reasons for low acceptance.

The VRESCo (Vienna Runtime Enviroment for Service-oriented Computing) project has the goal to provide an infrastructure to application developers that enables efficient and flexible development of service-oriented applications without bothering the developer with all the details of numerous Web service specifications such as UDDI [46]. It include services offered as public APIs for publishing, searching, register for notifications, service compositions and invocation of services. The VRESCo infrastructure will be explained in detail in Chapter 2.

The subject of this thesis is the composition component of VRESCo. The goal is to offer composition as a service functionality. That means that a composition of services can be created as an individual service within VRESCo. The focus lies on microflows. Microflows are simple workflows composed of several Web services without support of customer interaction at execution time.

To be able to offer this functionality three steps are required to be fulfilled:

- A way to define a microflow needs to be provided. Beside of functional dependencies and logical sequence in the service execution path the definition of QoS (Quality of Service) constraints should be supported. Moreover, the QoS constraints needs to support priority levels (e.g. required, hard, medium, weak).
- A service needs to resolve the modeled microflow by transforming it to a common workflow that can be executed on a common workflow engine and to choose the correct and best service implementations. QoS constraint hierar-

chies needs to be respected in this step.

- The finally transformed and composed workflow needs to be hosted as a service.

A DSL (Domain Specific Language) is a programming language for solving problems in a particular domain by providing built-in abstractions and notations for that domain. DSLs are usually small, more declarative than imperative, less expressive and more attractive than general-purpose languages because of easier programming, systematic reuse, better productivity, reliability, maintainability, and flexibility [42].

In this thesis a DSL named VCL (Vienna Composition Language) is introduced for microflow modeling. In this language, developers can specify what functional (e.g., inputs, outputs) and non-functional constraints (e.g., QoS) each service in a composition has to fulfill. QoS can be specified for the overall composition as well as for single services by using constraint hierarchies to express a fine-grained distinction of the importance of a constraint [59].

The modeled microflow within VCL needs to be interpreted. This process is not trivial. By using a data-flow driven way of expressing service links among each other a dependency tree needs to be constructed. Therefore the algorithm published in [19] can be used and extended. These extensions include support to alter the default data-flow behavior by adding control structures like while, if-then-else or wait to the mentioned algorithm. The modification on the dependency graph with the control structures will result in a composition graph. The composition graph represents the final composition that needs to be transformed into the target workflow engine. Windows Workflow Foundation [11] is used as the target workflow engine in our case.

Before processing the final composition graph, the QoS constraint hierarchy needs to be solved. By relying on the VRESCo as a service register it is required to find a service implementation for every service involved in the microflow. VRESCo can offer more than one candidate for a service. The functional constraints are fulfilled by each of them, they differ only by their QoS values. For every feature a service candidate is chosen based on the result of the solved constraint hierarchy. Thereby all required local and global QoS constraints needs to be satisfied. The number of satisfied optional constraint is maximized to optimize the quality of the composition.

Having finally a composition graph with selected service candidates the transformation into a Windows Workflow Foundation can be done. The difference to classical way of using WF is the fact that the WF workflow is created dynamically at composition runtime and not by a user at design time.

The last step is to publish the Web service to a Web server. As WF is used as a workflow engine the service will be hosted in the IIS (Internet Information Server).

1.2 Microflow Example

Microflows are fine-grained short-running composite services that do not allow user interactions at runtime. That means when a microflow is triggered, the caller waits for the result without the possibility to have any influence on the result. To illustrate a microflow an example will be introduced. The thesis will also reference to this microflow in the following chapters.

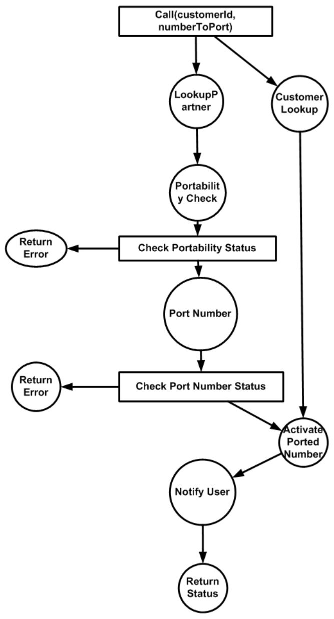


Figure 4: Telco Example

Figure 4 depicts a service composition for number porting. A mobile phone customer can port his phone number when changing the mobile operator. Therefore, one of the biggest disadvantage of changing the mobile provider for customers is gone.

To implement the Composition as a Service idea, the whole composition needs to be encapsulated into a standalone service. The composition has an entry point with two receiving parameters: `customerID` and the `number` that should be ported. After the service is called the caller can wait for the return value without the possibility to influence the execution. Therefore, this composition can be classified as a microflow.

By calling the microflow with the two parameters, the services *Lookup Partner* and *Customer Lookup* are invoked in parallel. Parallel execution is possible whenever a direct or indirect dependency is not present. The *Lookup Partner* service returns the name of the current mobile operator who owns the number that should be ported. The *Customer Lookup* service provides the typical customer data with name, address, email and so on.

The *Portability Check* service is used to test if the number can be ported at all. A number can, for example, be blocked by the owning mobile operator. Depending on the return value an error message will be returned or the microflow will continue with the *Port Number* service. On a success number porting the number will be activated on the new mobile operator by calling the *Activate Ported Number* service. This is the first place where a customer information from the *Customer Lookup* service is required. That means that the *Activate Ported Number* service will not be started until the *Customer Lookup* service returns a response and the previous port number check is positive.

After a successful number porting, the customer needs to be notified by an e-mail. As input the *Notify User* service requests customers e-mail and a status from the *Activate Ported Number* service. This is also a great point to show a direct and in-direct dataflow dependency. The e-mail address from the *Customer Lookup* service is a direct dependency for the *Notify User* service. Nevertheless, the dependency is not defined by a straight line between those two services. Because of the existing dependency between the *Customer Lookup* and *Activate Ported Number* service that appears before the *Notify User* is executed the dependency is in an in-direct way satisfied. Direct, in-direct and other service dependencies will be discussed in Chapter 3.

Additionally, non-functional constraints can be defined for the microflow, for example the response time can be constraint to a maximum of 1000ms or the availability must be higher than 95.5%. It is also possible to add optional or weighted constraints, where a non-fulfillment will not fail the microflow creation. Considering the fact that all services from the example can have more than one candidate with satisfied functional constraints. Therefore, a selection based on QoS values needs to be performed. Constraint hierarchies will be a topic in Chapter 3.

1.3 Organization of this Thesis

The rest of this thesis is organized as follows:

Chapter 2 will present the current state of the art and related work. An overview of the VRESCo Framework and its components will be given followed by a BPEL introduction. The chapter ends with an introduction into Domain-Specific Languages.

Chapter 3 is the main part of this thesis and will deal with all design decision of the proposed approach. It starts with design and syntax of the VCL DSL. The semantic interpretation and transformation of VLC to a structured composition will be covered. An important part is the QoS based optimization and service candidate selection. The chapter will be concluded with a description of the final transformation of the composition to a workflow runtime, in our case the Microsoft

Windows Workflow Foundation.

Chapter 4 depicts some implementation details. It follows the same order as Chapter 3 starting with the VCL implementation in MGrammar and ending with the publishing process of the generated WF workflows to the IIS.

Finally Chapter 5 will conclude the thesis with some final remarks and an overview of future work.

2 State of the Art and Related Work

This chapter will give an overview of related academic research and industrial standards to this thesis. Considering the fact that the implementation of this thesis is one of the core parts of the VRESCo framework it is natural to start with an overview of it. All core parts of VRESCo will be shortly introduced and the most important details will be explained. In the second part, an overview of current service composition languages will be given. The de-facto standard language BPEL will be introduced. The last part in this chapter will deal with DSLs, their definition and usage.

2.1 VRESCo

VRESCo (Vienna Runtime Environment for Service-Oriented Computing) aims at addressing some of the current challenges in Service-Oriented Computing research and practice. This includes topics related to service discovery and metadata, dynamic binding and invocation, service monitoring, QoS-aware service composition, service management and service notifications [44]. The VRESCo environment enables efficient and flexible development of service-oriented applications without forcing the developer to deal with all the details of numerous Web service specifications such as UDDI.

Unlike UDDI, the VRESCo runtime, acting as a service register, introduces an abstraction layer between the technical and business characteristics of a service. Two services with the same business intention can have different technical implementation. In case of two SOAP Web services this will lead to different WSDL files. Having different WSDL descriptions will lead to a problem for the software development. To exchange the service the whole consumer application needs to be modified and recompiled. The flexibility to change the service at runtime is far away. This is unfortunately the common situation in today's service-oriented solutions. To solve this limitation VRESCo introduced a feature-driven approach for service registration. A feature is a higher level abstraction of a service implementation. It defines common features of a service like their input and output parameters. A concrete service can be mapped to a feature. The detailed metamodel for this abstraction layer will be explained later in this chapter. By focusing the service consumer on features and not on service implementations, a decoupling effect can be reached. The Daios framework adds the required possibility to bind to a VRESCo feature. The binding is done on runtime, that means dynamic binding and invocation. Daios is also performing the required mediation from the feature level abstraction to the chosen service implementation. Having touched the basics of the VRESCo ideal an overview of the core parts will be given.

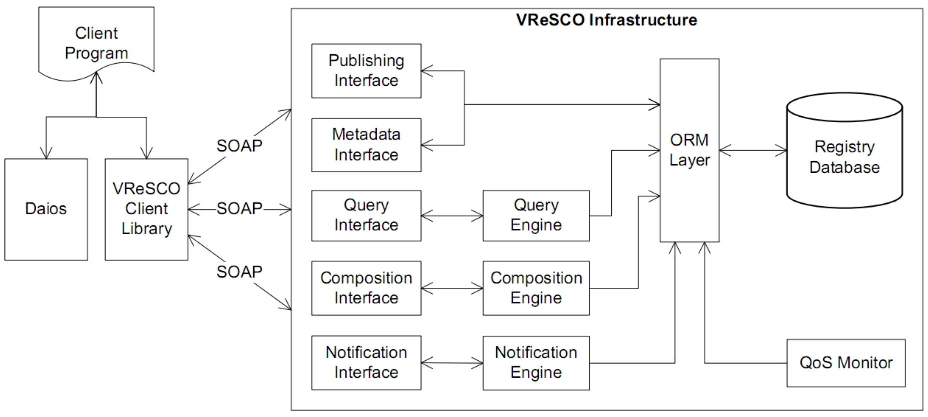


Figure 5: VRESCo Overview (from [59])

Figure 5 depicts the VRESCo system. All services within the VRESCo infrastructure are offered as SOAP Web services. The most important VRESCo services for application developers are:

- Metadata Service: The Metadata service offers all required functionality to manage the metadata of a service. Most important is the possibility to specify the mediation rules for the mapping between a service and a VRESCo feature. The feature itself is also manageable (i.e., create, modify, delete) over this service. Moreover, categories, data-concepts as well as QoS properties can be defined over the metadata service.

- Publishing Service: Unlike the metadata service the publishing service is used to manage concrete service implementations and not the related metadata. The main purpose of the service is to publish Web services to the VRESCo infrastructure. Two types are supported: static and dynamic publishing. Static publishing is done over a web frontend and dynamic publishing is done at runtime and makes a service immediately available for others within the infrastructure. There is also a versioning support (i.e., branching and merging revisions, tagging revisions) included that helps to avoid the classical drawbacks related to a service lifecycle. The versioning details will not be covered in this thesis, but they are very well described in [44].

- Searching and Querying Service: It allows to query and search for available services published within the infrastructure. Querying enables to find exact matches of the query string in service descriptions and its non-functional attributes (e.g., QoS), whereas searching allows to find services using full text queries with fuzzy matches. The query-language is similar to HQL (Hibernate Query Language) and easy to use. The composition service – the main focus of this thesis – will use the querying service heavily to get all service candidates for a requested feature in a given composition.

- Daios: Daios stands for Dynamic, Asynchronous and Message-oriented Invocation of Web Services and was developed by Philipp Leitner in his master's thesis [38]. Daios is a client-side Web service framework used to bind to features provided by VRESCo. Its important to mention that the binding is done dynamically and provides also the ability to rebind to different feature implementations (i.e., Web services) on runtime. The required mediation between feature and Web service is also performed in Daios. More details will be presented below.

- Notification Service: Notifications play a central role in SOAs, although there are not very explicit. This service allows the developer to register for receiving different events: i) when new services in specific categories are available, ii) when the QoS of a service changes below or above a given threshold, iii) when the service interface has changed. The use of notifications is transparent in the sense that the client library handles the creation of local endpoints to receive notifications from VRESCo if an event occurs.

- Composition Service: The Composition Service is the main focus of this thesis. The VRESCo infrastructure is used to achieve a composition of services by letting the user to specify composition requests in a domain specific language (DSL). A composition request consists of functional requirements for every feature like input and output parameters as well as non-functional requirements (e.g. QoS Values) and finally the composition of features. The information encoded in the DSL is used and matched with the services available within VRESCo. All concepts and details will be covered in later chapters. It is mentioned here just for the classification of the service within the VRESCo infrastructure.

- VRESCo Client Library: The VRESCo Client Library is a .NET Component to help developers to access the VRESCo core services. The component can be seen as a wrapper to the core components with a simple interface. It will be used to address the publishing, metadata and querying services within the compositions service.

2.1.1 VRESCo Metadata

By following the SOA idea in a large-scale enterprise environment a particular service can be offered by more than one service provider. By referring to the example presented in the first chapter the Notify Service is a great candidate for this. By assuming that the Notify Service is sending a SMS to the customer, a so called SMS Gateway is required. Only a short query on the Internet gives a huge amount of SMS Gateways like Clicktell or T-Mobile. Besides the different QoS values like response time, availability or price, also the technical interface is not equal. By generating service stubs from provided WSDL the calling method will look different for every service. This is shown in Listing 1.

```
//First SMS service proxy
SMS1Client sms1Service = new SMS1Client();

//First SMS service proxy
SMS2Client sms2Service = new SMS2Client();

//required parameter
string smsText = "";
string sendTo = "+4369912915301";
string sendFrom = "+4369912915302";

//response
bool status;

//send sms by using the first provider
status = sms1Service.SendSMS(smsText, sendTo, sendFrom);

//create request object for second provider
SmsRequest request = new SmsRequest(){Message = smsText, To = sendTo, From
    = sendFrom};
SmsResponse response = SMS2Client.Send(request);
status = response.Status;
```

Listing 1: SMS Services

SMS1Client and SMS2Client are created from the WSDL of two given SMS services. The offered functionality is the same, both services are sending a SMS to a destination number with a given message. But from a technical point of view they are different. The first SMS service has a method called `SendSMS` and three string parameters. The second service is using simply the word `Send` for the method and takes one complex parameter that consists of all three strings that are provided in the first sample. The response types are also different, the first service is dealing with a Boolean value where the second service encapsulates the Boolean value in a complex type. Both approaches for dealing with parameters are widely spread. By using a complex type for input and output parameters a change can be made to them by adding new encapsulated properties without having a breaking change in the service. For a developer it is not possible to create a generic client application that talk with SMS1Service and SMS2Service, and is able to use a third service in the future. The application is so called hard-wired. The decision for a service needs to be made in the design time of an application. This is of course a big disadvantage. It can be easily possible that a new provider with a cheaper price per SMS will appear. The software needs to be modified for the new service because of the different endpoint address and WSDL. In most cases the modifications on the software is not the only problem, depending on the type of the software and the deployment scenario, a huge number of software clients needs to be updated with the new version.

This is a common scenario in today SOA development. The main reason for this problem is located on a technical and not on business level. To avoid this limitation an abstraction layer is introduced to hide the technical details. The client software is build and bind against an abstract Service, and not a concrete service. The concrete service can be exchanged transparently to the client. A redeployment and update of the client software is no longer needed.

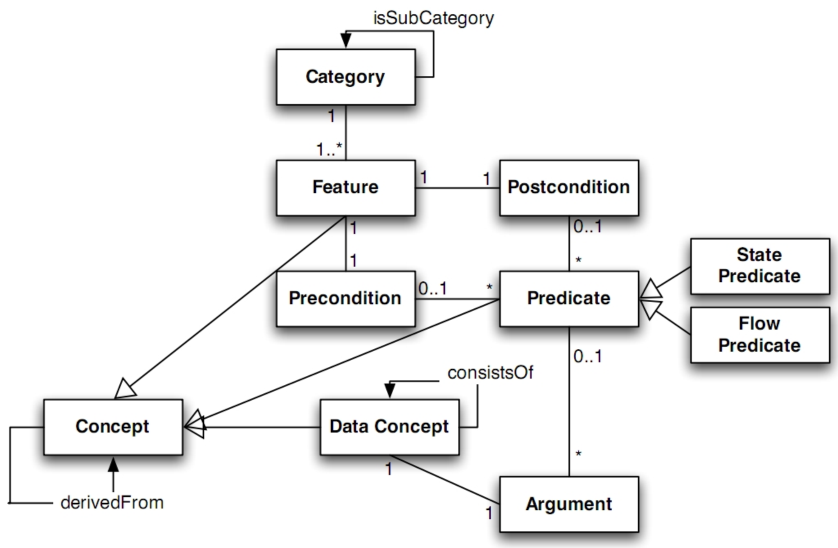


Figure 6: VRESCo Metadata Overview (from [62])

Figure 6 depicts the metadata model included in VRESCo that facilitate the abstraction layer. The following description of the metadata model is taken partially from Rosenberg et al.[62]. The main building blocks of the VRESCo metadata model are Concepts. A Concept has a specific meaning in a given domain. There are three different types of concepts (inherits from Concept):

- Features represent activities in the domain (e.g. send SMS)
- State Concepts represent states in the domain (e.g. user notified)
- Data Concepts represent concrete entities in the domain (e.g. User or SMS text)

A Data Concept can contains other Data Concepts. For example an address Data Concept can consists of a street, zip, city, state and country. Therefore we distinguish between simple and complex Data Concepts. A simple Data Concept is based on a predefined type like String or Integer. On the contrary a complex Data Concept contains at least one simple or complex Data Concept. Therefore, a simple Data Concept would be the string based country Data Concept and the address

Data Concept containing all the simple Data Concepts mentioned above would be complex.

Feature Concepts are associated with one Category to be easily classified. A Category can contain also subcategories. The behavior of a sub-category is similar to object oriented inheritance, all Features defined for the parent category are also available for the sub-category. Each Feature can have a Precondition and a Postcondition. They consists of logical statements that needs to be fulfilled before, respectively after the service execution. Both conditions are composed of multiple Predicates that can have a number of Arguments. A Predicate can be divided into one of this:

- Flow Predicate: Is related to the data flow of a concept. By using the keywords requires(X) and produces(X) where X is a concept a data flow dependency can be created. In our `Notify User` scenario we would have a Feature `Send` and the Data Concepts `SMSMessage` and `SMSResult`. The precondition would have the predicate requires(SMSMessage) and the postcondition the predicate produces(SMSResult).
- State Predicate: Are used to specify a global condition that is valid before (Precondition) or after (Postcondition) a feature is executed.

2.1.2 Mapping VRESCo Metamodel to Service Layer

A service that is registered into the VRESCo Infrastructure needs to be mapped to the explained metamodel in the last section. Figure 7 depicts the mapping rules between the domain metamodel and a concrete service.

A concrete service is grouped into Categories. A Category on the other hand can have more than one Service. Service operations are mapped to Features and a service parameters to Data Concepts. Features and Data Concepts can be mapped to more than one operation respectively parameter. This leads to the possibility to have more than one service abstracted to a specific domain metamodel. A service grouped into a specific Category needs to provide at least a set of operations that can be mapped to all existing Features within the Category. If a service is assigned to more than one Category, all included Features need to have their counterpart on the operation side. The Post and Pre State constrains for a service can be mapped to the Pre and Postcoditions on the domain metadata layer. It is important to mention that operation and parameters of a service are well defined in the provided WSDL file. On the other hand Post and Pre States are not covered and supported within WSDL. The following listings will show the code for mapping a service to the domain model by using the provided VRESCo Client Library. The code is used

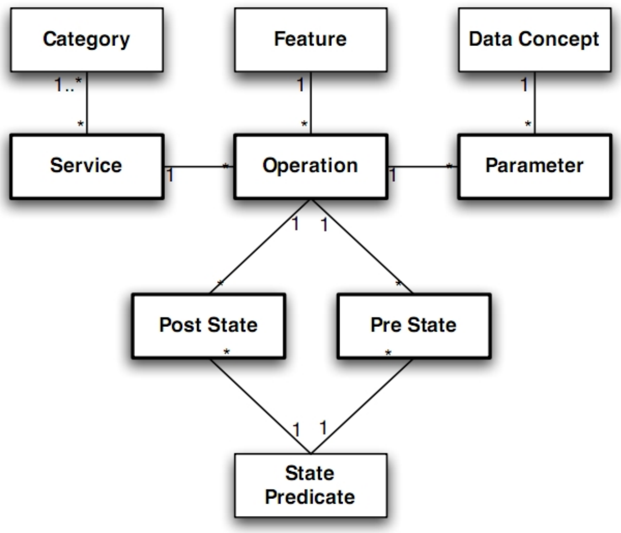


Figure 7: VRESCo Metadata Mapping (from [62])

within the example composition explained in the first chapter to register a simple SMSService into VRESCo.

```
private static IVReSCOPublisher publisher =
  VReSCOClientFactory.CreatePublisher("guest");

private static IVReSCOMetadataPublisher metadatapublisher =
  VReSCOClientFactory.CreateMetaDataPublisher("guest");
```

Listing 2: Create a VRESCo Publisher and MetaDataPublisher

To create a domain within the metamodel , register and map a service in VRESCo a proxy for the publisher and metadata service is required. The proxy can be easily created with the Client Library (Listing 2).

The first step is to create a new domain model with a category and the appropriate features and data concepts. Listing 3 shows how to use the metadata service to create a NotificationService category with a NotifyUser feature and the corresponding data concept for the input parameter. The output parameter and data concept are hidden in the listing because the implementation is similar to the input parameter respectively data concept. Listing 4 is the registration itself done with the publishing service.

A service revision is required for the concrete service. The publisher service offers the `CreateNewService` and `AddOperation` methods to register a service revision respectively an operation. There is also a `SetOperationQoS` method that is used to

```
ServiceCategory notifyCategory = new ServiceCategory("NotificationService")
    ;
notifyCategory = metadatapublisher.CreateCategory(notifyCategory);

Feature notifyFeature = new Feature();
notifyFeature.Name = "NotifyUser";

DataConcept notifyInput = new DataConcept();
notifyInput.Name = "NotifyRequest";
notifyInput = metadatapublisher.CreateDataConcept(notifyInput);
notifyInput = metadatapublisher.AddSubElement(notifyInput,
 DataConcepts.String, "Number", 1, 1);
notifyInput = metadatapublisher.AddSubElement(notifyInput,
 DataConcepts.String, "Message", 1, 1);

Parameter paramNotifyIn = new Parameter();
paramNotifyIn.DataConcept = notifyInput;
paramNotifyIn.Name = "request";
paramNotifyIn.MaxOccurs = 1;
paramNotifyIn.MinOccurs = 1;
paramNotifyIn.IsOutParameter = false;

Parameter paramNotifyOut = new Parameter();
//.... similar to paramNotifyOut in

notifyCategory = metadatapublisher.AddFeature(notifyCategory, notifyFeature
    ,
                 new Parameter[2] { paramNotifyIn, paramNotifyOut }, null)
                     ;
```

Listing 3: Register Feature and DataConcept in VRESCo

```
Service notificationServiceC1 = new Service();
notificationServiceC1.Name = "NotificationService";
ServiceRevision notificationServiceC1Rev = new ServiceRevision();
notificationServiceC1Rev.Service = notificationServiceC1;
notificationServiceC1Rev.Wsdl = "http://localhost:12011/NotificationService
    ?wsdl";
notificationServiceC1Rev.Contract = "INotificationService";
notificationServiceC1Rev = publisher.CreateNewService(notificationServiceC1
    ,
 notifyCategory, user, notificationServiceC1Rev);
Operation notificationC1Op = new Operation();
notificationC1Op.Name = "NotifyUser";
Parameter notificationC1Request = new Parameter();
notificationC1Request.Name = "request";
notificationC1Request.DataConcept = notifyInput;
notificationC1Request.MaxOccurs = 1;
notificationC1Request.MinOccurs = 1;
notificationC1Request.IsOutParameter = false;
notificationServiceC1Rev = publisher.AddOperation(notificationServiceC1Rev.
    Id,
   notificationC1Op, new Parameter[2] { notificationC1Request,
       notificationC1Response }, null, notifyCategory.Features[0]);
```

Listing 4: Register a Service VRESCo

set QoS values for a particular service revision. The last part is the mapping of a service revision to operation. This is shown in the Listing 5.

```
Mapper notifyc1Mapper = metadatapublisher.CreateMapper(notifyCategory.
    Features[0],
 notificationServiceC1Rev.Operations[0]);

Assign notifyc1RequestAssign = new Assign(notifyc1Mapper.
    FeatureInputParameters[0],
 notifyc1Mapper.OperationInputParameters[0]);
notifyc1Mapper.AddFeatureToOperationFunction(notifyc1RequestAssign);

metadatapublisher.AddMapping(notifyc1Mapper.GetMapping());
```

Listing 5: Add a Mapping

The Mapper object can be obtained from the metadata service by providing the feature and the operation that should be mapped. An Assign object expresses a direct mapping between the parameter in the domain model and the service parameter. In this sample it is an easy mapping because the structure of the both parameters are equal. A complex mapping is also possible within VRESCo but will not be discussed in this thesis. The details are explained in the [62] paper. The last statement calls the AddMapping method and adds the created mapping.

2.1.3 DAIOS (Dynamic, Asynchronous and Message-oriented Invocation of Web Services)

DAIOS is a .NET Client Component that is used to consume features published in the VRESCo environment. The general idea of DAIOS is to provide a stubless way to use services and hide a much as possible of the technical implementation. The whole communication is done with DAIOS messages that are transformed into a format which is used natively by the service. Currently it supports WSDL and REST typed services. Additionally, there is also a VRESCo mediator included, which is used to transform features calls to service operations, and data concepts to parameter and vice versa. Thereby, a mechanism is offered to dynamically bind and invoke service. It is important to mentioned the core functionality at this point because it will be used in later chapters for dynamic invocation of services composed into a composition. For more details the published paper [37] is a good reference.

2.1.4 VRESCo Querying Service

The last topic in the VRESCo overview part is the provided querying service. By using the metadata and publishing service a domain model and appropriate services can be added to VRESCo. The second element in the “Publish-Find-Bind” principal is the ability to get a service with specific characteristics from a service registry instance. VRESCo has the VQL (VRESCo Query Language) for this purpose. It provides the functionality to query for all information stored in the VRESCo registry database. The syntax is similar to HQL, a query language that is used in the widely spread OR mapper Hibernate. Listing 6 shows a typical VQL query.

```
var query = new VQuery(typeof(VReSCO.Contracts.Core.ServiceRevision));

query.Add(Expression.Eq("IsActive", true));
query.Add(Expression.Eq("Service.Category.Name ", "NotificationService"));
query.Add(Expression.Eq("Operations.Feature.Name", "NotifyUser"));
query.Match(Expression.And(
                    Expression.Eq("Operations.QoS.Property.Name", Constants.
                        QOS_PRICE),
                    Expression.Le("Operations.QoS.DoubleValue", maxPrice)));

var strategy = new ExactQuerying(maxResults);
return strategy.Query<VReSCO.Contracts.Core.ServiceRevision>(query,
    NHibernateContext.Session);
```

Listing 6: VQL Query

The constructor of the VQuery takes the wanted result type as parameter. The query is defined by adding criteria to the query object. A criteria contains one or more expressions. An expression supports logical(e.g., AND,OR,...) and equivalent operators(e.g., greater than, less or equal, ...). There is also a support for querying strategies. For this thesis only the exact strategy is relevant, which is used to get only results that fulfill all defined criteria. There are also priority and relaxed querying strategies where all criteria's does not have to be satisfied. The Query method returns finally a collection of result elements, in this case a collection of ServiceRevisions.

2.2 Composition Languages

Web services facilitate complete new business-to-business and enterprise integration models. Built on defined standards like WSDL and SOAP an integration and communication over the company boundaries is easily possible. It is no more required to have all IT and Domain know-how in one company. A good example is the SMS Getaway service. Today, a software with SMS notifications can be easily developed. Architects and developers can focus on the domain and core part of the application without dealing with SMS notifications details like contracts with and between individual mobile network providers. Small companies, on the other side, can offer a specialized Web service as a product. The know-how is only for the SMS system required, the cost structure and the service quality is much better because of the specialization level.

On the other hand, big companies have a huge amount of Web services in their own infrastructure. There is also a tendency to offer legacy systems over a Web service interface inside the company to increase software reusability, commonly known as EAI(Enterprise Application Integration)[25]. In case of simple business cases where just a small number of Web services are involved often an own software part is developed that is using the services. But having huge business workflows that need to be mapped to the service level is a big challenge. Often the root business workflow can change, and the change itself needs to be implemented on the service level as soon as possible. Composition languages have been introduced to deal with these problems. A composition language is used map a business workflow to a service composition. The composition is based on classical control structures and the invocation of the services. It specifies the execution path of particular Web services as well as their input and output parameters. Decisions constructs are included to support handling of different workflows states and error scenarios within the composition. The goal is to specify a service composition model that is compiled and executed on a composition engine.

BPEL (Business Process Execution Language) [34] is the most used composition language for Web service composition. It was introduced in 2002 by Microsoft and IBM by borrowing language features from Microsoft's XLANG [66] and IBM's WSFL [39]. BPEL built on top of Web service standards WSDL and SOAP. Like both of them, BPEL compositions are written in XML. The language itself is complex and unreadable. Often designers with a GUI are used to create a BPEL composition. A good overview of the XML syntax and the collaboration with WSDL is given in [68].

A BPEL composition, or to use the correct wording, a BPEL process specification is similar to a flow-chart. It is composed of activities. Activities are divided into primitive or structured. Primitive activities are:

- `invoke`: invokes a specific Web service operation described in WSDL
- `receive`: waits for a message from an external source
- `reply`: replies with a message to an external source
- `wait`: go into an idle state for a given period
- `assign`: copy data object called data containers from one to another
- `throw`: notifies that an error occurs within the execution
- `terminate`: terminate the whole service instance

- `empty`: empty block, do nothing

Structured activities are:

- `sequence`: defines an execution order
- `switch`: conditional statement to select an execution path based on a condition
- `while`: offers looping
- `pick`: is used for race conditions based on timing or external triggers
- `flow`: is used to support parallel execution
- `scope`: is used to group activities

In the paper [74] an evaluation of supported workflow patterns is done. There is also a comparison table of supported patterns within other composition languages like XLANG, WSFL, BPML and WSCI.

The general problems of BPEL are covered within the two mention papers, [68] and [74]. Nevertheless, there are some other problems that we identified within BPEL. First of all it is impossible to abstract the service layer from the business layer because of a strict binding to WSDL. Two alternative services need to implement exactly the same interface. This is of course a problem, especially if the service is located outside of company boundaries. The language itself is XML based and tends to grow disproportionate with the complexity of the composition. Another topic is QoS-awareness, it is generally not supported. Most of this topics will be addressed within the composition approach presented in this thesis.

2.3 Domain Specific Languages

A typical programming language like C or Java is used to solve many different problems. A developer needs to have the know-how to program in this language and understand the problem domain. Therefore, this languages can be classified as general purpose or generic languages because of their widely spread usage. Alternatively, there are also domain-specific languages that are limited to solve a specific problem within a defined domain. Good examples are HTML and CSS, or BPEL mentioned in the last section. Such a language can be classified as a specific language in a domain. In some cases it is hard to decide if a language is generic or specific. This is a the case with Cobol and Fortran. Cobol is used for business processing and Fortran for scientific numeric computations. A typical domain is specified in both cases. But both languages are not restricted to these domains, and can be used to solve other problems. Therefore, Cobol as well as Fortran are typical generic languages with an influence of a specific domain. In general there are three different approaches to split the domain from the general language presented in [70]:

- Subroutine libraries contain subroutines that perform related tasks in well-defined domains like, for instance, differential equations, graphics, user-interfaces and databases. The subroutine library is the classical method for packaging reusable domain-knowledge.
- Object-oriented frameworks and component frameworks continue the idea of subroutine libraries. Classical libraries have a flat structure, and the application invokes the library. In object-oriented frameworks it is often the case that the framework is in control, and invokes methods provided by the application-specific code.
- A domain-specific language is a small, usually declarative, language that offers expressive power focused on a particular problem domain. In many cases, DSL programs are translated to calls to a common subroutine library and the DSL can be viewed as a means to hide the details of that library.

Van Deursen et al. proposed also the following definition for a DSL:

A domain-specifc language (DSL) is a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain.

DSLs allow us to map a domain and it boundaries to the language. Domain experts can use the DSL without the problem to work with a complex programming language. The abstraction layer can be defined on a level that fits perfectly to the given domain. The domain related code, workflow or problem definition and solution is located on one location without the drawback to have it distributed over thousands of lines of code. Therefore, DSLs are enhancing the productivity, reliability, maintainability and portability.

But there are of course disadvantages related to DSLs. To design a DSL is not an easy task and requires a lot of domain and implementation knowledge. To find the right scope for the DSL and to balance between domain-specific and general-purpose language constructs can be a challenging task. Additionally, there are costs for designing, implementation and maintenance. DSL users have to be educated to work with the DSL.

3 Composition as a Service

This chapter presents our proposed Composition as a Service (CaaS) approach. It starts with a short overview of the whole system. The second section will introduce the Vienna Composition Language (VCL) for composition modeling. The transformation from a semantic model to a structured composition is the topic of the third section. The fourth section will deal with QoS constraint hierarchies and the their resolution. The last section will describe the transformation of the structured composition to a workflow instance for a common workflow engine and their deployment.

3.1 Overview

The Software as a Service approach has become an important and powerful SOA fundamental in the last few years. Software reusability and maintainability has grown within and over company boundaries. One of the main benefits of the SaaS approach is the ability to compose new functionality out of existing services into so-called “composite services”, thus significantly increasing reusability of existing services. But the composition or orchestration task is not a trivial. The higher the number of involved services within a composition ,the harder it is to manage and hold the overview over the composition.

Currently, there are two approaches to create a service composition. The classical way is to create an application or service in a common programming language that is referencing and using all included services. This, even though it is the most common way, has a lot of disadvantages. Besides the change management problems, there is the whole overhead of hosting the application within the infrastructure which needs to be done manually.

BPEL process descriptions follows the second approach to deal with service compositions. A service composition, or so-called process description in BPEL, is described in a XML format and deployed to a composition engine for execution. In contrast to the first approach, the hosting task is completely covered by the composition engine. Nevertheless, both approaches are hard-wired to WSDL and make it not possible to replace services with other similar services. BPEL offers at least UDDI support to cover the “Publish-Find-Bind” SOA principal. But having the included services hard-wired to WSDL descriptions, which is also the case if UDDI is used, lowers the flexibility to change service providers dynamically if a better service becomes available. The reason can be a better QoS value for the offered response time or a better pricing offer. At least there are diverse modeling tools with a graphic interface which are helping to hide the XML complex for the user.

This thesis introduces a Composition as a Service (CaaS) approach to deal with the problems described above. Relying on the VRESCO infrastructure and using it as a service register the problem with the “Publish-Find-Bind” principal is solved by building a composition against VRESCO features and not concrete service interface. To offer a simple and expressive way to design compositions the domain specific language VCL is introduced. VCL is able to define constraint hierarchies on QoS values like response time, availability or security. To avoid over-constraint compositions, constraint hierarchies offer optional constraints with a strength level that makes it possible to publish a composition even if not all constraints are fulfilled. The core component, the composition service within VRESCO, is used to interpret the defined composition in VCL, create the composition, deploy it automatically and offer it immediately as a Web service.

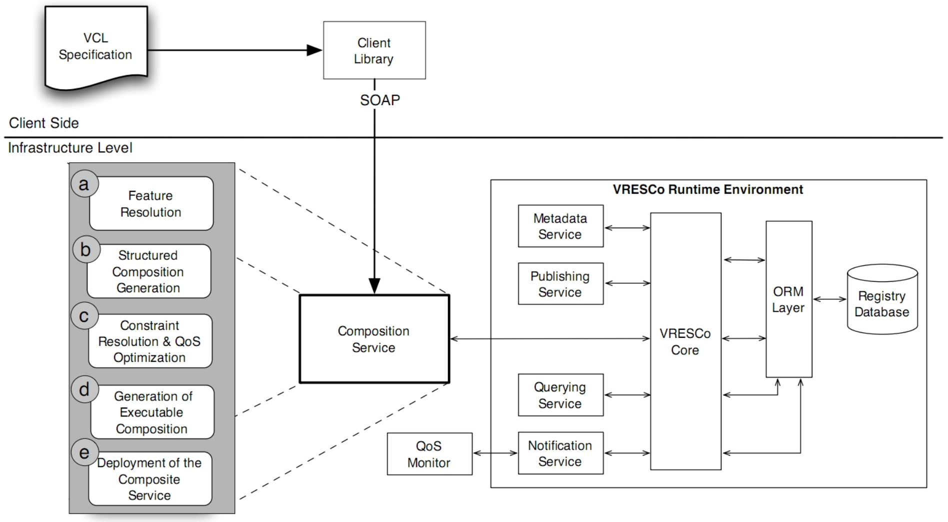


Figure 8: Composition as a Service (from [61])

Figure 8 depicts an overview of the developed composition infrastructure. The VCL specification is a written service composition in VCL. The service client is a .NET DLL that is used to interpret the VCL specification, check if syntax errors are included and transform it to a semantic model. The semantic model is passed to the composition service within the VRESCO infrastructure.

At first, the VRESCO service registry will be queried for required features and service candidates. If a feature without a service candidate is found the composition can’t be created and the client will get an error. The next step is to create a structured composition. This is done by analyising the given VCL semantic model with a transfomation alorithm which creates an execution graph for the composition. Other than BPEL, the exact execution order has not to be complete within the VCL specification. By using a mix of defined control structures within the VCL specification and an implicit data-flow dependency model a concrete execution order is derived.

VCL supports mandatory and optional constraints on feature and composition level. That means an optional response time constraint can be defied on the whole composition, so-called global constraints, and a required availability constraint on a particular feature within the composition, so-called local constraints. This leads to a complex constraint hierarchy problem that is transformed to an optimization problem and solved with an optimization solver. At this point a VCL specification can be over-constrained and the user needs to relax some constraints to be able to deploy the composition successfully.

The next step is to transform the structured composition, with selected service candidates as a result from the constraint hierarchy, into a workflow for a common workflow engine. Windows Workflow Foundation is used as a target workflow engine because of the ability to deploy and host is easily achieved within the Internet Information Server. The last step is the deployment itself, where the WF workflow is deployed as a Web service. The user gets a status message with the WSDL location for the new deployed service.

3.2 Vienna Composition Language

The Vienna Composition Language is a domain specific language with the goal to provide an intuitive and simple DSL for the purpose of service composition within the VRESCO infrastructure. Because of the VRESCO abstraction layer, the user is modeling a composition out of features and not concrete services. This design decision provides the ability to select dynamically the best service candidate based on functional and non-functional characteristics [61].

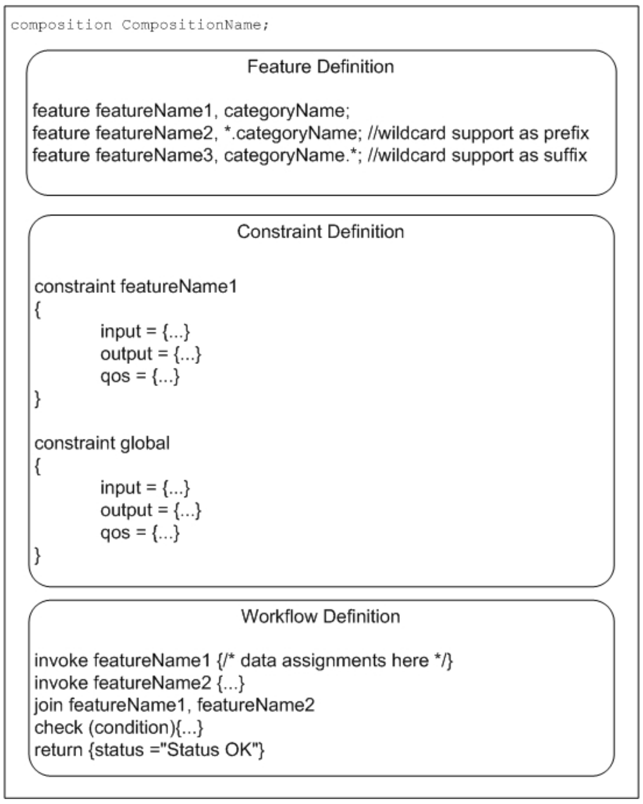
A typical VCL composition consists of three sections a) a list of all involved features within the composition b) a section with functional and non-functional constraints and c) a description of the execution order or workflow.

A VCL composition starts with the keyword `composition` followed by the name of the composition and the a semicolon. In general, the VCL syntax has borrowed a lot from the C language. We decide to take this syntax style because of the wide spread languages like C#, C++ or Java which are familiar to many developers. For our Telco sample the first line is shown in Listing 7.

```
composition TelcoCasestudyComposition;
```

Listing 7: Composition Name Definition

In the feature definition section all used features need to be specified that are involved in the composition. Listing 8 shows the definition for the Telco sample. The keyword `feature` is used to indicate a beginning feature definition statement followed by the



```
composition CompositionName;

feature featureName1, categoryName;
feature featureName2, *.categoryName; //wildcard support as prefix
feature featureName3, categoryName.*; //wildcard support as suffix

constraint featureName1
{
        input = {...}
        output = {...}
        qos = {...}
}

constraint global
{
        input = {...}
        output = {...}
        qos = {...}
}

invoke featureName1 {/* data assignments here */}
invoke featureName2 {...}
join featureName1, featureName2
check (condition){...}
return {status ="Status OK"}
```

Figure 9: VCL Overview

feature name and category and finished again with a semicolon. It is important to note that the feature name is not the name of a concrete service but the name a feature within the VRESCO service registry. A feature can be constraint to a specific service candidate within the constraint definition section. There is also a support for prefix and suffix wildcards within the category name to simplify the support of sub-categories.

The constraint section can be divided into global constraint section and local constraint sections for individual features. The order within the VCL composition is not important.

```
feature Crm, *.CustomerService.CustomerLookup;
feature LookupPartner, *.PhoneManagementService.LookupPartner;
feature PortCheck, *.PortingService.PortabilityCheck;
feature PortNumber, *.PortingService.PortNumber;
feature ActivatePort, *.PhoneManagementService.ActivatePortedNumber;
feature Notify, *.NotificationService.NotifyUser;
```

Listing 8: Used Feature List

```
constraint global
{
  input = {
    int customerId;
    string numberToPort;
  }
  output = {
    string status;
  }
  qos = {
    responseTime = 7000;
    availability = 0.66;
  }
}
```

Listing 9: Global Constraints

A constraint definition element always starts with the `constraint` keyword followed by the name of the constraint. For global constraints, always the word `global` is used and for local constraints the name or the feature. Within a constraint element functional and non-functional constraints can be specified. Functional constraints are inputs and outputs. They will start with `input =` respectively `output =` and have their content surrounded with curly brackets. The content itself is a representation of a VRESCO Data Concept. That means for a local constraint that the given feature has also to have the specific input and output data concept registered within VRESCO. In case of global constraint, an input and output constraint is used to specify the interface and in the end the WSDL of the resulting composition. The global constraint given in Listing 9 will result in a composed Web service with a method that is taking a `customerId` and a `numberToPort` as input parameter and returning a status. It is important to note that all input and output constraints are strongly typed. That means for the global constraint that the `customerId` is an integer and the parameters `numberToPort` and status are simple string. VCL also supports complex types. This is shown in the following listing.

In this sample the local constraint for the Crm feature is using complex types as input and output constraints. The CustomerLookupRequest is in this case a complex type that consists of the integer CustomerId. The CustomerLookupResponse is more complicated and contains the typical customer data set.

The non-functional constraints are located behind the `qos` keyword. For this thesis and the first version of VCL only a subset of common quality of service values are supported. But the language can be easily extended to support every other quality of service attribute that is included within the VRESCO environment. All `qos` elements fits into the `qosName = value;` pattern. It is important to know that the equal character does not mean that the response time needs to be exactly a given value but rather depending on the context need to be greater, equal or lower that

```
constraint Crm
{
  input = {
    CustomerLookupRequest[
      int CustomerId;
    ]
  }

  output = {
    CustomerLookupResponse[
      string Firstname;
      string Lastname;
      string PhoneNumber;
      string Mail;
      string Street;
      string Zip;
      string City;
    ]
  }

  qos = {
    responseTime = 1000;
    availability = 0.66;
  }
}
```

Listing 10: Local Constraints

the given value. The value itself is also not strong type because of typical values types that are used for different quality of service types. For example the response time is always given as an integer, whereas the availability is declared as a double. Response time and the availability are also a good sample for values with different meaning of the equals symbol. The constraint value on response time means that the concrete service value needs to be lower that the constraint value on the other hand an availability has to be higher that the given value. The following list will give an overview of supported QoS attributes by VCL:

- Response Time is given in msec. A constraint of max. 5 seconds will be `responseTime = 5000`
- Availability is constraint by the percentage. The value is given as a decimal value. `availiability = 99,95` means an availability of 99,95 or higher is required
- Reliable Message is a typical Boolean typed constraint. The only two possible values are `true` or `false`.
- Security is enumeration typed constraint. VCL supports X509, None, UsernamePassword and IntegratedSecurity.

- Accuracy is like availability constraint with a percentage value. The value is also a decimal value and means that the feature require at least the constraint value.
- Throughput is given as an integer that stands for invocations per second. To satisfy the constraint the feature needs to provide the constraint number of invocations per second or more.
- Price is expressed with an integer and defines the maximum cost for a single invocation.

There is also a special constraint type named `service`. It is used to select a service candidate for a feature directly. This means also that this feature has a fixed service candidate and the feature itself is not a target of the constraint hierarchy optimization. To be more precise, the QoS values are of course included in the calculation of global QoS values, but there is not a service candidate selection done within the optimization process even if there is a service candidate with better QoS values. Listing 11 shows such a constraint.

```
constraint FeatureName
{
  serivce = {
    serviceName= MasterCardWebService;
  }
}
```

Listing 11: Service Name Constraints

The last part of a VCL specification is the workflow definition section. The VCL user needs to specify relationships between features and their input respectively output values. With additional control structures and implicit data-flow rules the target composition is specified.

VCL supports an amount of control structure statements that will be listed and explained in this section. First of all it is necessary to explain the data-flow principle. To do this, the control structure `invoke` needs to be introduced.

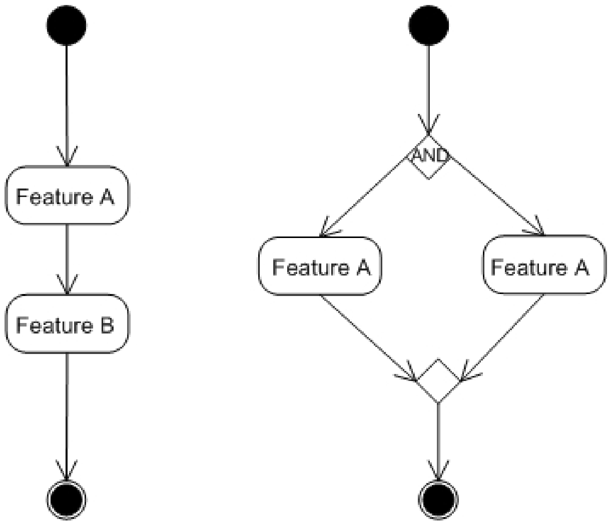
```
invoke PortCheck {
  PortabilityCheckRequest[
    NumberToPort = numberToPort;
    NewProvider = LookupPartner.LookupPartnerResponse.ProviderName;
  ]
}
```

Listing 12: Used Data Concepts by the PortCheck Feature

The `invoke` statement is used to call a feature. Therefore, the feature name is placed after `invoke` followed by a section for input parameter specification. In the constraint definition section the `PortCheck` feature is defined with an input constraint which contains a complex type `ProtabilityCheckRequest` containing a `NumberToPort` and `NewProvider` member. In the `invoke` statement, values needs to be assigned to the input parameter. In this case, this is done in Line 3 and 4 in Listing 12. The important part is located on the right side of the equal operator. `numberToPort` was defined within the global constraint as an input parameter. Therefore, the parameter can be used as an input value within features input constraints. The `LookupPartnerResponse` type is defined as an output constraint for the `LookupPartner` feature. After a feature is called, the output parameter can be used within an input for other features.

To use an output parameter from `Feature A` as an input parameter for `Feature B`, `Feature A` needs to be executed before `Feature B`, because there is an existing data dependency.

```
invoke Feature A...
invoke Feature B...
```



The workflow definition segment can have two meaning a) invoke first `Feature A` and afterwards `Feature B` b) invoke `Feature A` and `Feature B` in parallel. VCL will interpret this example as sequence of (`Feature A`, `Feature B`) if `Feature B` has a data dependency to `Feature A`, otherwise both features will be executed in parallel.

In general, all invoke statements are invoked in parallel if there is no data-flow dependency or an explicit control structure. This has the advantage to optimize the composition automatically and take this task from the developer. On the other hand, the developer needs to be aware that the features will be executed in parallel.

A control structure that can break the data-flow rule to execute in parallel is the sync statement. It is used for synchronization within a composition.

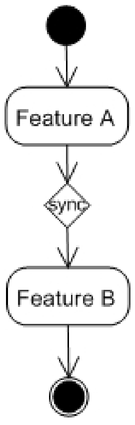


Figure 10: VCL Sync

In Figure 10 `Feature B` is explicitly executed after `Feature A` independently from data-flow dependencies between them. It is necessary to note that there is a floating root node. All invoked features have an implicit dependency to the current root-node. The current root node is at the beginning of a composition the entry point itself. By using control structures like sync it is moved to the sync node. This behavior is also included within the figure 10. `Feature A` has the entry point as a root node. The sync statement is above the `Feature B` invoke statement. Therefore, the root node is changed to the sync node and derived to the `Feature B`.

The next important control structure is the `while` statement. It used to support loops in VCL.

```
while [4](payment.PaymentResponse.status != OK) {
  invoke p1;
}
```

Listing 13: While Statement

The while is defined with an expression. As long as the expression is true the body of the while statement is looped. A while statement is, similar to the sync statement, also becoming a root-node. But different to sync, it is only valid within the while body and note outside. This is required to hold all invocations within a while and not to move them out. Without the root node an invoked feature that has no data dependencies to other features within the while would be outside of the root node. Another optional information that can be added to a while statement is the estimated number of loops (e.g., [4]). This is used within the QoS optimization algorithm for a better approximation. This value is entered as an integer enclosed by square brackets after the keyword `while`.

Conditional statements are fundamental constructs in nearly every language. This is also valid for VCL (Listing 14). The conditional statement is implemented with the `check` and optional `else` statement in VCL. The check statement has a mandatory expression that needs to be fulfilled to execute the content block. Nested check statements are supported. The else statement is optional and has also optional probability value that is included in front of the content body enclosed by square brackets. The probability value is used later in the QoS optimization and is expressed with an interval from 0.0 to 1.0.

```
check (PortCheck.PortabilityCheckResponse.IsPortable = 1)
{
  invoke PortNumber {
    PortNumberRequest[
      NumberToPort = numberToPort;
    ]
  }
}
else [0.1]
{
  throw "Number can't be ported by external provider";
}
```

Listing 14: Conditional Statement

Error handling is implemented by the keyword `throw` and an error message (Listing 15). The throw statement returns immediately the error message to the composition caller. In general, it is used in a combination with a conditional statement to check return values of particular features and identify if the composition can continue with the execution or a state is reached where the composition has to stop.

```
throw "Number can't be ported by external provider";
```

Listing 15: Throw Statement

The keyword supported by VCL is `return`. It is used to confirm a successful composition call and to set the returning data object (Listing 16).

```
return {
  status = "Job done";
}
```

Listing 16: Return Statement

3.2.1 Evaluating VCL's Workflow Pattern Support

Van der Aalst presents in [67] a set of 20 workflow patterns that can be used to classify a composition or workflow language based on their expression power. He also evaluates the most widely spread composition languages in [74] and gives an overview in a table with supported patterns for each language. This section will describe shortly all supported patterns and show if and how they are supported by VCL. At the end, the table provided by [74] will be extended with a VCL column. The short description of each patter are cited directly from [67].

WP1 Sequence: An activity in a workflow process is enabled after the completion of another activity in the same process.

```
invoke feature1 ...
invoke feature2 ... //only if feature2 is using return values as parameters
      from feature1

//OR explicit with sync

invoke feature1 ...
sync feature1;
invoke feature2 ...
```

Listing 17: WP1 Sequence

As mentioned in previous section, a sequence can be realized with a data-flow dependency or an explicit sync statement.

WP2 Parallel Split: A point in the workflow process where a single thread of control splits into multiple threads of control which can be executed in parallel, thus allowing activities to be executed simultaneously or in any order.

```
invoke feature1 ...
invoke feature2 ... //only if feature2 has no data-flow dependency to
      feature1
```

Listing 18: WP2 Parallel Split

A parallel split is the standard behavior if two or more features are invoked in a row and no data-flow dependency exists.

WP3 Synchronization: A point in the workflow process where multiple parallel subprocesses/activities converge into one single thread of control, thus synchronizing multiple threads. It is an assumption of this pattern that each incoming branch of a synchronizer is executed only once.

```
invoke feature1 ...
invoke feature2 ... //only if feature2 has no data−flow dependency to
    feature1
invoke feature3 ...
sync feature1 , feature2 ;
```

Listing 19: WP3 Synchronization

The `sync` statement is used to synchronize two or more features. The composition will continue to execute when both features calls are done. In the given example `feature1` and `feature2` are synchronized, `feature3` will run in parallel to the `sync`.

WP4 Exclusive Choice: A point in the workflow process where, based on a decision or workflow control data, one of several branches is chosen.

```
check(expression) {
  invoke feature1
  ...
}
else {
  invoke feature2
  ...
}
```

Listing 20: WP4 Exclusive Choice

The exclusive choice pattern is implemented with the `check` statement. Based on the expression value the `check` or e`else` branch is chosen.

WP5 Simple Merge: A point in the workflow process where two or more alternative branches come together without synchronization. It is an assumption of this pattern that none of the alternative branches is ever executed in parallel.

```
invoke feature1 ...
invoke feature2 ...
check(feature1.status=ok || feature2.status=ok)
{
  invoke feature3 ...
}
```

Listing 21: WP5 Simple Merge

A simple merge pattern can be realized by using check statement that verifies status values from both called features. This works because a `check` statement is implicitly also a `sync` statement for all involved features within the expression.

WP6 Multi-choice: A point in the workflow process where, based on a decision or workflow control data, a number of branches are chosen.

```
invoke feature1 ...
invoke feature2 ...
check(feature1.status=ok) {
  invoke feature3
  ...
}
check(feature2.status=ok) {
  invoke feature3
  ...
}
```

Listing 22: WP6 Multi-choice

The Multi-choice pattern is not supported natively by VCL . But it can be achieved by using a `check` statement for every branch individually.

WP7 Synchronizing Merge: A point in the workflow process where multiple paths converge into one single thread. If more than one path is taken, synchronization of the active threads needs to take place. If only one path is taken, the alternative branches should reconverge without synchronization. It is an assumption of this pattern that a branch that has already been activated, cannot be activated again while the merge is still waiting for other branches to complete.

```
invoke feature1 ...
invoke feature2 ... //only if feature2 has no data-flow dependency to
    feature1
invoke feature3 ... // no data-flow dependency to feature1 or feature2
sync feature1 , feature2 , feature3;
```

Listing 23: WP7 Synchronizing Merge

This pattern is similar to the Synchronization pattern. The difference is that more than two features or branches are synchronized.

WP8 Multi-merge: A point in a workflow process where two or more branches reconverge without synchronization. If more than one branch gets activated, possibly concurrently, the activity following the merge is started for every activation of every incoming branch.

Multi-merge is not supported natively. The only way to call the same feature after one or more branches executed is to check the state of every branch or feature and call the required feature directly. It the example above the invoke `feature3` and `invoke4` are not synchronized. `feature3` is synchronized only with `feature1` and `feature4` with `feature2`.

```
invoke feature1 ...
invoke feature2 ... //only if feature2 has no data-flow dependency to
    feature1
check(feature1.status=OK) {
  invoke  feature3
  ...;
}
check(feature2.status=OK) {
  invoke  feature4
  ...;
}
```

Listing 24: WP8 Multi-merge

WP9 Discriminator: The discriminator is a point in a workflow process that waits for one of the incoming branches to complete before activating the subsequent activity. From that moment on it waits for all remaining branches to complete and ignores them. Once all incoming branches have been triggered, it resets itself so that it can be triggered again (which is important otherwise it could not really be used in the context of a loop).

The Discriminator pattern is not supported in VCL. There are two reasons for not being able to have a "first-come, first-served" behavior a) VCL doesn't support any state variables beside the response values of called features and b) an expression that needs to be evaluate within a while or check statement always requires all features involved in the expression to be synchronized.

WP10 Arbitrary Cycles: A point in a workflow process where one or more activities can be done repeatedly.

```
while(expression) {
  invoke feature1 ...;
  invoke feature1 ...;
}
```

Listing 25: WP10 Arbitrary Cycles

Loops are natively supported by VCL with the `while` statement.

WP11 Implicit Termination: A given sub-process should be terminated when there is nothing else to be done. In other words, there are no active activities in the workflow and no other activity can be made active (and at the same time the workflow is not in deadlock).

An explicit `return` statement is not required in VCL. But this is only meaningful if the composition has no return object.

```
invoke feature1;
invoke feature1;
```

Listing 26: WP11 Implicit Termination

Patterns 12 to 15 are patterns that involves multiple instances of a feature. Multiple instances have only limited support in VCL. A feature can only be called multiple time within a loop. For this reason none of these patterns is supported.

Also patterns from 15 to 20 are not supported because there are related to state values within a composition and external inputs. A triggered VCL composition cannot be changed by an external input. In general, VCL composition are microflows as described in the introduction chapter. They are, opposite to BPEL, short running without external interfaces that can influence their execution. VCL also doesn't support variables to hold interim values. Therefore these patterns are also not supported.

Table 1 extents the table from [67] and gives an overview and comparison to other composition languages.

Pattern name	**VCL**	**BPEL**	**XLANG**	**WSFL**	**BPML**	**WSCI**
Sequence	+	+	+	+	+	+
Parallel Split	+	+	+	+	+	+
Synchronization	+	+	+	+	+	+
Exclusive Choice	+	+	+	+	+	+
Simple Merge	+	+	+	+	+	+
Multi Choice	+	+	-	+	-	-
Synchronizing Merge	+	+	-	+	-	-
Multi-Merge	+	-	-	-	+/-	+/-
Discriminator	-	-	-	-	-	-
Arbitrary Cycles	+	-	-	-	-	-
Implicit Termination	+	+	-	+	+	+
MI without Synchronization	-	+	+	+	+	+
MI with a Priori Design Time Knowledge	-	+	+	+	+	+
MI with a Priori Runtime Knowledge	-	-	-	-	-	-
MI without a Priori Runtime Knowledge	-	-	-	-	-	-
Deferred Choice	-	+	+	-	+	+
Interleaved Parallel Routing	-	+/-	-	-	-	-
Milestone	-	-	-	-	-	-
Cancel Activity	-	+	+	+	+	+
Cancel Case	-	+	+	+	+	+

Table 1: Supported Workflow Patterns

3.3 Abstract Dependency Graph and Structured Composition

A VCL composition is transformed by the client library to a semantic model, and forwarded to VRESCOs composition service. The first task in the composition service is to analyze the model and create a structured composition that can be used for further operations and optimizations.

BPEL describes a composition with a strict and explicit language. An alternative approach is to use a graph-based composition by analyzing input/output dependencies between services or features. In a graph-based composition services are coordinated through control elements like AND-splits, AND-joins, XOR-splits and XOR-joins. A disadvantage of this approach is that such a graph can contain flaws, for example deadlocks or cycles [19].

This problem with graph-based models disappears if the models are structured, i.e., if each split has a corresponding join and if the split-join pairs are properly nested. Models violating this constraint are similar to programs containing GOTO statements. A BPEL composition is mainly structured [19].

Our approach is a mix of pure data-flow dependencies enriched with a small set of control structures within VCL to support an automatic annotation of XOR and AND splits and joins. Furthermore, there is also a simple control structure for loops. The concrete syntax of the VCL is described in the previous section.

The provided semantic model from the VCL parser contains only a set of features and control statements. A validation is also not done at this moment. To solve this the semantic model is transformed in a structured composition. A structured model is desired and necessary for our approach also based on the following reasons[59]:

- it enables enactment of a structured composition on existing composition or workflow engines, thus, removing need to implement an execution engine for VCL
- it allows to detect flaws in the unstructured composition such as deadlocks which may lead to runtime errors at a later stage
- it facilitates an efficient QoS aggregation based on well-know workflow and composition patterns

To get a structured composition we use an algorithm introduced in [19] by Eshuis and modified it to fulfill our requirements. This algorithm creates a composition structure based on data-flow dependencies. It consists of three steps:

- Create an Abstract Dependency Graph: An abstract graph is an adjacent list of features and their dependencies to each other.
- Create a Structured Composition: The structured composition is created by applying the provided algorithm on the abstract graph.

- Annotate Nodes: Annotation of control flow decisions in the structured composition with either AND (executed in parallel) or XOR (conditional execution).

Eshuis et al. introduced the algorithm as a semi-automated approach. The first two steps can be automated, but the third step needs additional user input. This is covered in our approach with the additional control structure supported by the VCL. The original algorithm is designed to create a structured composition only based on a set of services and its input and output types.

In our approach, the user specifies within the VCL explicit the workflow and uses data flow dependencies implicitly. Therefore, the original algorithm is slightly modify. The main modifications are done in the first step by adding additional nodes to the abstract dependency graph. The second step is implemented without any modification on the original algorithm. The last step, annotations of the nodes, is done partially within the ADG creation and partially after the structured composition algorithm is done. The concrete modifications will be explained in detail in the following subsections.

3.3.1 Abstract Dependency Graph

Features communicate, like services, through instances of data concepts or messages in the service terminology. A data concept can be primitive or consists of a set of child data concepts. Given a data concept m, we denote by $\mathit{type}(m)$ the set of child data concepts of the data concept m and m itself. For each Feature f, $\mathit{input}(f)$ denotes its input data concept and $\mathit{output}(f)$ its output data concept.

Based on the input/output data types of each feature, we can define dependencies between features. If a feature A outputs a data concept and feature B needs as input this data concept, than B depends on A.

Dependencies between a set S of services are captured in a graph. An abstract dependency graph is a tuple (SE) with [19]:

$S \overset{\text{df}}{=} \{s_1, s_2, \cdots, s_n\}$ a set of services

$E \overset{\text{df}}{=} \{(s, s') \in S \times S \mid \mathit{type}\left(\mathit{data}\left(\mathit{output}\left(s\right)\right)\right) \cap \mathit{type}\left(\mathit{data}\left(\mathit{output}\left(s'\right)\right)\right) \neq \emptyset\}$

The following two helper functions are defined on the abstract dependency graph and will be used later for the structured composition algorithm. Given a feature s, its set of pre-condition features, written $\mathit{pre}(s)$ are those features on which s depends. Symmetrically, the set of post condition features of s, written $\mathit{post}(s)$, are those features that depend on s [19]:

$pre\,(s) \overset{\text{df}}{=} \{x \mid (x,y) \in E \wedge y = s\}$

$pre\,(s) \overset{\text{df}}{=} \{y \mid (x,y) \in E \wedge x = s\}$

There are two constraints that needs to be satisfied by the abstract dependency graph:

- The dependency graph is acyclic.
- If there is an edge from s1 to s2, then there is no path with length greater than 1 from s1 to s2.

The first constraint is required to forbid loops. Eshuis et al. left the relaxation of this constraint for future work. Our approach adds support for loops by adding a dummy node in the ADG with a `LOOP` annotation. This will be explained later.

The second constraint is a requirement for the structured composition algorithm. Figure 11 depicts a sample for this problem.

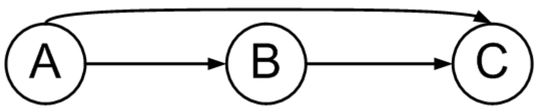


Figure 11: ADG Dependency Length

Feature C depends on A and B. This can be easily repaired by either removing the violating dependency since it is redundant, or by putting an dummy feature. The last solution is used to implement if-then-else structures with an empty else branch.

In the depicted problem the first solution would be to remove the edge between A and C. This is show on Figure 12.

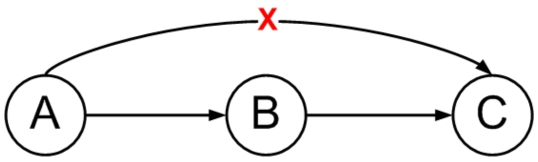


Figure 12: Remove ADG Violation Dependency

The second solution is to include a dummy node that maps a xor statement. Image 13 shows this solution.

In the original algorithm, provided by Eshuis et al., this is one of the points where the user needs to decide. In our solution this is not required because we get this information's already from the VCL specification. If there is a XOR relation, covered by the `check` VCL keyword, the second case is used otherwise the additional relation

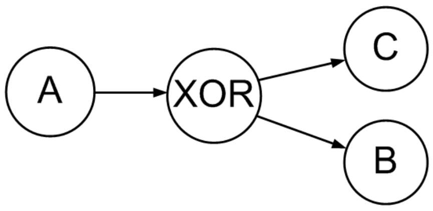


Figure 13: Add Dummy Node to Map a XOR Statement

is added to the adjacent list. At the end of the ADG creation an iteration is done on the ADG that removes all violating dependencies by removing the direct nodes as proposed in the first solution.

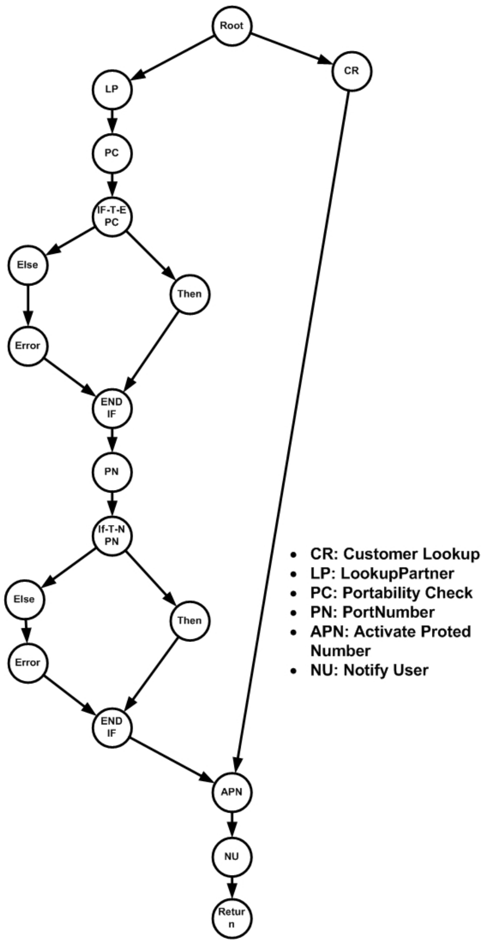


Figure 14: ADG for TELCO Example

Image 14 depicts a graphical representation of the ADG for the TELCO Example presented in the introduction section. This will be used to explain all required changes on the original algorithm to fit to our VCL solution.

The first modification is the root element on Figure 15. The goal of our Composition as Service approach is to have a standalone Web service that contains the whole composition. Therefore VCL contains the global feature, that is used as an entry point to the composition. At the beginning of the composition the entry node is also the current root node. But, mentioned already in the previous section, the current root node is floating. All invoked features have an implicit dependency to the current root-node. By using control structures like `sync` or `check` it is moved to this node. This behavior can be seen on the `PortNumber` feature. Until the first `IF-T-E` statement the current root node is the entry point itself. After `IF-T-E` the root node is change to the closing dummy node `ENDIF`.

An `IF-T-E` statement is implemented by four dummy features. This is used to implement later a simple annotation of the structured composition with `XOR` and `AND` attributes. The first feature is `IF-T-E`, followed by `ELSE` and `THEN` features and ending with the `ENDIF` feature. The `ELSE` and `THEN` Feature have a direct dependency to the `IF-T-E` feature. The `ENDIF` feature on the other side has a dependency to all features included in the `ELSE` and `THEN` features and the features itself. This is required to hold the `IF-T-E` construct in balance, otherwise some features without a data dependency, but a specified control structure within the VCL script would be outside of the construct.

Within the `THEN` and `ELSE` futures an internal root node is used. In general, a `THEN` feature should not be seen as a feature, it is more a block construct similar to classic programming languages. Therefore an own root node is used for the content (e.g., included features). The initial value of the internal root node is the `THEN` dummy feature itself respectively `ELSE`. The reason for this requirement is again the merge of a pure data-flow composition approach and strict control structures. Of course, the internal root node behavior is recursive. A `THEN` feature can contain again `IF-T-E` constructs.

The global or most outer root node is, as explained, at the beginning of the ADG creations set to the entry point feature. By using the control structures (e.g., check, sync and while) the root node will be moved. This is small disadvantage for our approach. A feature without any data-dependencies that is called in the VCL after a control structure will have a dependency to the this control structure and not to the entry point. This behavior can be easily avoided by moving the call of the feature in front of the control structure.

Loops are supported by the while control structure. An abstract dependency graph is shown on Figure 15.

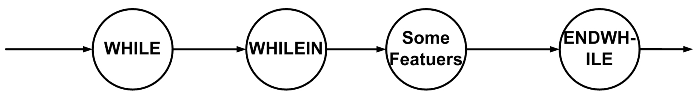


Figure 15: ADG Loop Example

The while statement is similar to the check statement. It consists of three dummy features. The `WHILE` feature is the outer feature that denote a loop, the `WHILEIN` is the inner feature like the `THEN` feature. It is also the initial value for the internal root node. The `ENDWHILE` feature is the enclosing dummy feature for the while statement and acts as a new global root node value. Like `ENDIF`, all internal features have a dependency to them.

The sync statement is implemented by a single dummy feature that has a direct dependency to features that are specified to be synchronized and is used as the current value of the root node. Picture 16 depicts an example.

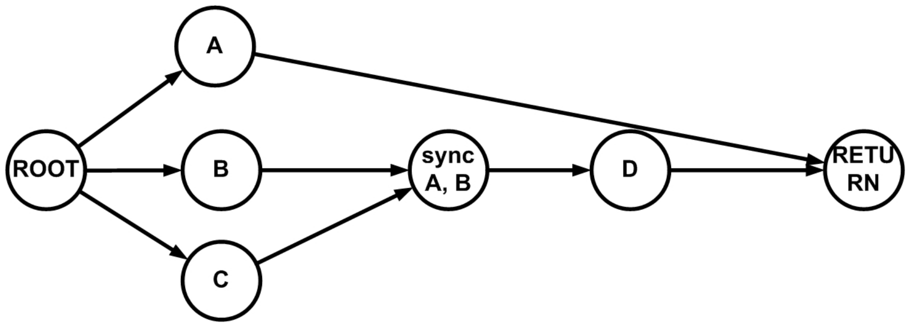


Figure 16: ADG Sync Example

Feature `B`, `C` are synchronized in the `sync` feature. This is done by adding a direct dependency between `sync` and `B` respectively `sync` and `C`. The feature `A` is not synchronized because it is called before the sync statement and is not part of it. The change of the current root node to the `sync` feature is also not affecting the `A` feature, whereas feature `D` has now a direct dependency to `sync`.

3.3.2 Structured Composition

A well-defined ADG is the input for the structured composition algorithm. A structured composition can be formalized to a hierarchical view, where leaf nodes are services (e.g., features) and non-leaf nodes are blocks. In the graphical syntax, the beginning and end of a block is demarcated by a split and join node respective [19].

There are two types of blocks, composite blocks of type `COMP` and sequential blocks of type `SEQ`. In the original algorithm a `COMP` block is annotated as `AND` or `XOR`. This list extended by the `WHILE` annotation that behaves exactly like `AND` but is used as an additional information in the constraint resolving and publishing task.

The children of blocks are specified as parameters, a set in case of `COMP` and a list in case of `SEQ` blocks. For example `COMPSEQ[X,Y],SEQ[Z]` is a process in which `X` is done before `Y` and both are done in parallel with or exclusive to `Z` [19]. The following definition is given by Eshuis:

Given a set of S of services, the following inductive definition formalizes the set of structured compositions on S:

- Each service s ∈ S is a structured composition.
- If X1, X2,...,Xn are structured compositions, then so are SEQ[X1, X2,...,Xn] and COMPX1,X2,...,Xn.

Figure 17 depicts the graphical syntax of TELCOs structured composition. Every block has its corresponding closing block node. Figure 17 shows also the added control structures to the ADG very well. An If-Then-Else statement is transformed to `IF-T-E`, `ThenBlock`, `ElseBlock` and `Sync` nodes. A feature is always nested within a SEQ block. This is not mandatory, an example is the SEQ-CR-SEQ block. Seq is redundant in this case, but having it there is not a performance issue but simplifies the algorithm implementation.

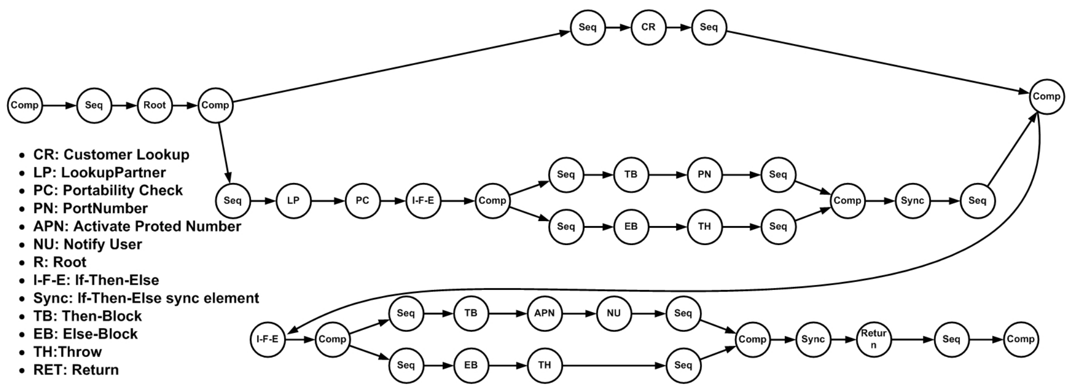


Figure 17: Strucutred Composition for the TELCO Example

The same structured composition is show with in a different representation in Listing 27.

The algorithm introduced by Eshuis et al. is shown as pseudo-code in Listing 28. It takes a dependency graph as input and returns a valid structured composition. The algorithm starts with an initial composition and extends its iteratively. the initial composition is created in Line 2. The Initial(S,E) procedure returns a list of features without dependencies to other features. Considering the modifications of the ADG performed in the previous step, this list contains always the Root feature because all other features depend directly or indirectly on it. Nevertheless, the Initial procedure can be specified as:

$$Initial\,(S,E) = \{s_1 \in S \mid \nexists s_2 \in S : (s_1, s_2) \in E\}$$

```
{
  COMP {
    SEQ [ROOT,
    COMP{
      SEQ [Crm], SEQ[LookupPartner, PortCheck, IfThenBlock,
      COMP{
         SEQ[ThenBlock, PortNumber], SEQ [ElseBlock, Throw]
      }
      , Sync ]
    }
    , IfThenBlock,
    COMP{
      SEQ[ThenBlock, ActivatePort, Notify], SEQ[ElseBlock, Throw]
    }
    , Sync, Return]
  }
}
```

Listing 27: Structured Composition for Telco Example

```
procedure STRUCTUREDCOMPOSITION((S,E))
  C:=SEQ[constructBlock(Initial(S,E))]
  processed:=Initial(S,E)
  while processed ≠ S do do
    toprocess := next(processed)
    for each maximal influencing subset I of toprocess do
      BlockI := constructBlock(I)
      InputI := {s ∈ processed|post(s) ∈ I}
      N:= the most nested block in C containing all services in InputI.
      if N is composit then
        NotPreI := {c ∈ children(N)|InputI ∩ services(c) = ∅}
        if NotPreI ≠ ∅ then
          PreI := COMP{c ∈ children(N)|InputI ∩ services(c) ≠ ∅}
          N' := COMP({SEQ[PreI, BlockI]} ∪ NotPreI)
          replace N by N' in C
        else
          parent(N).append(BlockI)
        end if
      else
        parent(N).append(BlockI)
      end if
      processed := processed ∪ I.
    end for
  end while
  return C
end procedure
```

Listing 28: Structured Composition Algorithm

The constructBlock(X) function is a helper function that creates from a given set of features a single service or a composite block consisting of a set of sequential blocks, each containing one service from X [19]. The function is defined as:

$$constructBlock\,(X) = \begin{cases} x & \text{, if } X = \{x\} \\ COMP\,\{SEQ[x] \mid x \in X\} & \text{, otherwise} \end{cases}$$

Line 3 adds the set of the initial features (e.g. Root feature in our implementation) to the processed set. The main part of the algorithm is done in the while loop that runs until all features are included in the final composition.

On every loop the next function (Line 5) is called to decide which features should be performed in this iteration. This is the case for all unprocessed features whose pre-conditional features have been processed:

$$next\,(processed, S) \overset{\mathrm{df}}{=} \{s \in S \mid pre\,(s) \subseteq processed\} \setminus processed.$$

The complexity of the algorithm is added in the lines 6-23. The reason is that the set of features from toprocess set cannot be easily attached to the composition. Eshuis et al. depicts this problem on the Figure 18.

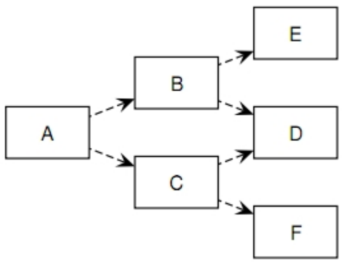


Figure 18: Service Grouping with Mutual Influence

Suppose features A,B and C have been processed, then next returns D, E, and F. Now, D depends on both B and C. To translate this into control flow, the blocks encompassing both B and C has to end before D. But this implies that the block also ends before E and F. To achieve this, D, E, and F need to be processed as a group [19].

To define precisely which services need to be processed in a group, we introduce the notion of influence. Two services are directly influenced by each other if they depend on the same feature, i.e., their pre-conditions overlap. In Figure 4, features B and C both depend on A, and therefore directly influence each other. Two services s1, s2 influence each other if they either directly influence each other or if another service s that directly influences s1 and influences s2 (e.g. the relation is transitive). For example, in Figure 4 features E and F influence each other even though their pre-conditions are disjoint, since D directly influences both E and F [19].

Therefore, the composition produced in one iteration needs, depending on the influenced features sets for this iteration, to be completely refactored. This is performed in lines 6-23. The source code of the implementation can be found in the appendix. Table 2 shows the iterations with the toprocess set and the interim composition for the Telco example.

To process	Composition
ROOT	COMP{SEQ[ROOT]}
LookupPartner	COMP{SEQ[ROOT,COMP{SEQ[Crm],SEQ[LookupPartner]}]}
PortCheck	COMP{SEQ[ROOT,COMP{SEQ[Crm],SEQ[LookupPartner,PortCheck]}]}
IfThenBlock	COMP{SEQ[ROOT,COMP{SEQ[Crm],SEQ[LookupPartner,PortCheck, IfThenBlock]}]}
ElseBlock	COMP{SEQ[ROOT,COMP{SEQ[Crm],SEQ[LookupPartner,PortCheck, IfThenBlock,COMP{SEQ[ThenBlock],SEQ[ElseBlock]}]}]}
Throw	COMP{SEQ[ROOT,COMP{SEQ[Crm],SEQ[LookupPartner,PortCheck, IfThenBlock,COMP{SEQ[ThenBlock,PortNumber],SEQ[ElseBlock,Throw]}]}]}
Sync	COMP{SEQ[ROOT,COMP{SEQ[Crm],SEQ[LookupPartner,PortCheck, IfThenBlock,COMP{SEQ[ThenBlock,PortNumber],SEQ[ElseBlock,Throw]},Sync]}]}
IfThenBlock	COMP{SEQ[ROOT,COMP{SEQ[Crm],SEQ[LookupPartner,PortCheck, IfThenBlock,COMP{SEQ[ThenBlock,PortNumber],SEQ[ElseBlock,Throw]},Sync]}, IfThenBlock]}
ElseBlock	COMP{SEQ[ROOT,COMP{SEQ[Crm],SEQ[LookupPartner,PortCheck,IfThenBlock, COMP{SEQ[ThenBlock,PortNumber],SEQ[ElseBlock,Throw]},Sync]},IfThenBlock,COMP{ SEQ[ThenBlock],SEQ[ElseBlock]}]}
Throw	COMP{SEQ[ROOT,COMP{SEQ[Crm],SEQ[LookupPartner,PortCheck,IfThenBlock ,COMP{SEQ[ThenBlock,PortNumber],SEQ[ElseBlock,Throw]},Sync]},IfThenBlock,COMP{ SEQ[ThenBlock,ActivatePort],SEQ[ElseBlock,Throw]}]}
Notify	COMP{SEQ[ROOT,COMP{SEQ[Crm],SEQ[LookupPartner,PortCheck,IfThenBlock, COMP{SEQ[ThenBlock,PortNumber],SEQ[ElseBlock,Throw]},Sync]},IfThenBlock, COMP{SEQ[ThenBlock,ActivatePort,Notify],SEQ[ElseBlock,Throw]}]}
Sync	COMP{SEQ[ROOT,COMP{SEQ[Crm],SEQ[LookupPartner,PortCheck,IfThenBlock, COMP{SEQ[ThenBlock,PortNumber],SEQ[ElseBlock,Throw]},Sync]},IfThenBlock, COMP{SEQ[ThenBlock,ActivatePort,Notify],SEQ[ElseBlock,Throw]},Sync]}
Return	COMP{SEQ[ROOT,COMP{SEQ[Crm],SEQ[LookupPartner,PortCheck,IfThenBlock, COMP{SEQ[ThenBlock,PortNumber],SEQ[ElseBlock,Throw]},Sync]},IfThenBlock, COMP{SEQ[ThenBlock,ActivatePort,Notify],SEQ[ElseBlock,Throw]},Sync,Return]}

Table 2: Composition Steps

The last step that need to be done to get the final structured composition is to perform the annotation of the COMP nodes with AND or XOR . Looking at the Figure 17 the general rule can be applied. If a COMP node depends on an If-Then-Else feature and a Then and Else feature depends on this COMP node than the COMP node is annotated with XOR, otherwise AND. The Figure 19 shows the same composition just with the final COMP annotations.

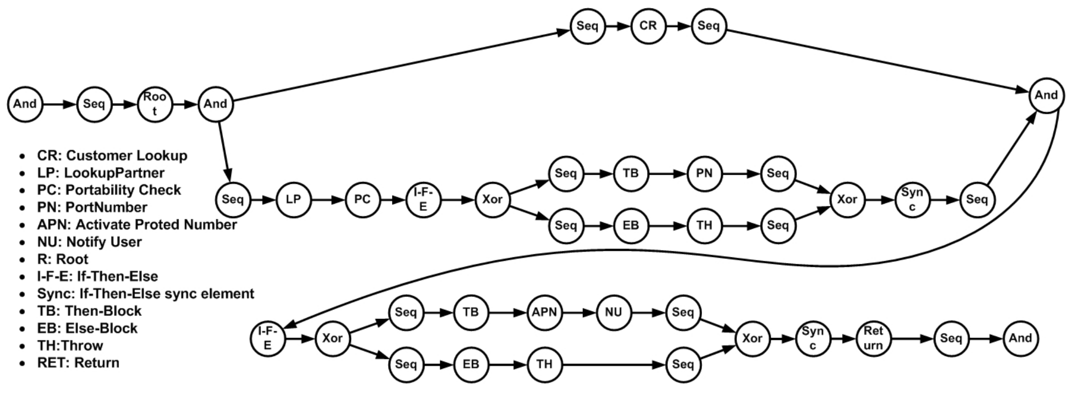


Figure 19: Telco Structured Composition with AND and XOR Annotations

3.4 VRESCO Querying for Service Candidates

Having a valid structured composition for a given VCL Script the composition service calls the VRESCO Query service, introduced in Chapter 2, to get for each feature a list of their service candidates. An example VQL query for the *CustomerLookup* feature given in Listing 29.

```
1  var query = new VQuery(typeof(VReSCO.Contracts.Core.ServiceRevision));
2
3  query.Add(Expression.Eq("IsActive", true));
4  query.Add(Expression.Eq("Service.Category.Name ", "CustomerService"));
5  query.Add(Expression.Eq("Operations.Feature.Name", "CustromerLookup"));
6
7  query.Match(Expression.And(
8                          Expression.Eq("Operations.Feature.Parameters.
                               DataConcept.Name", "CustomerLookupRequest"),
9                          Expression.Eq("Operations.Feature.Parameters.
                               IsOutParameter", false)));
10
11 query.Match(Expression.And(
12                         Expression.Eq("Operations.Feature.Parameters.
                               DataConcept.Name", "CustomerLookupResponse"),
13                         Expression.Eq("Operations.Feature.Parameters.
                               IsOutParameter", true)));
14
15 query.Match(Expression.And(
16                      Expression.Eq("Operations.QoS.Property.Name", Constants.
                            QOS_RESPONSE_TIME),
17                      Expression.Le("Operations.QoS.DoubleValue", 1000.00)));
18
19 query.Match(Expression.And(
20                      Expression.Eq("Operations.QoS.Property.Name", Constants.
                            QOS_AVAILABILITY),
21                      Expression.Le("Operations.QoS.DoubleValue", 0.66)));
22
23 var strategy = new ExactQuerying(maxResults);
24 var queryResults = strategy.Query<VReSCO.Contracts.Core.ServiceRevision>(
       query, NHibernateContext.Session);
```

Listing 29: VQL Query for CustomerLookup Example

The VRESCO category is constraint to CustomerService in Line 4. The feature name is set to CustomerLookup. It is also important to get only active service candidates, this is done in Line 3. The input and output data concepts are given in Lines 7-14. Lines 15-21 set a filter on the response time respectively availability of the service candidate. A QoS filter is only applied if the given VCL QOS constraint is local (e.g. feature constraint) and mandatory. Thereby the number of service candidates can be reduced and give a performance boost in the following QoS optimization step. Non-mandatory constraint cannot be pre-filtered because they are optional and the decision if a certain service candidate is selected is done in the QoS-hierarchy optimization. This is also the case for global constraints, both optional and mandatory.

3.5 QoS Constraint Hierarchies and their Optimization

When the structured composition is created for a given VCL script and every specified feature has at least one service candidate the QoS constraint problem need to be solved to select the best service candidate for each feature respecting global and local constraints.

In general, VCL supports global and local constraints. A local constraint can also be called feature constraint and is valid only for the given feature on which it is specified. A global constraint on the other side is valid for the whole composition. Therefore a global constraint is affected by every single service candidate selection.

Moreover, VCL allows the user to give a strength value to a constraint. This is known as a constraint hierarchy. VCL supports required, strong, medium and weak strength values. This list can be easily extended. The idea behind constraint hierarchies is avoid over-constraint system and to give the user the flexibility to express nice to have QoS values.

The four mentioned strength values can be divided in two groups, optional and required. Constraints that are required must be full field by the selected service candidates. If a required constraint can't be satisfied the composition service returns a message that indicated that the composition is over-constraint and stops the execution.

The composition service, to be more precise the constraint solver, tries to satisfy, besides the required constraints, as much as possible optional constraints. We introduced a scoring system to support the different strength values of optional constraints. A satisfied weak constraints gets 5 points, a medium 10 and a strong constraint 20 points. Therefore the constraint solver tries to maximize the total score for the whole composition to get the best service candidates selection. We choose a simple static scoring system. An improvement would be to have different scores for global and local constraints or to have a dynamic scoring systems that depends on the total number of features involved in the composition.

Figure 20 depicts a simple composition example that will be used to demonstrate the influence of QoS values and their constraints on the selected service candidates. Listing 30 gives a number of constraints that should be satisfied for within the composition. Finally, Table 3 contains a list of service candidates for each feature.

The composition consists of three features. F1 is executed first. When F1 returns a value a XOR split follows by calling exclusively F2 or F3. To simplify the example

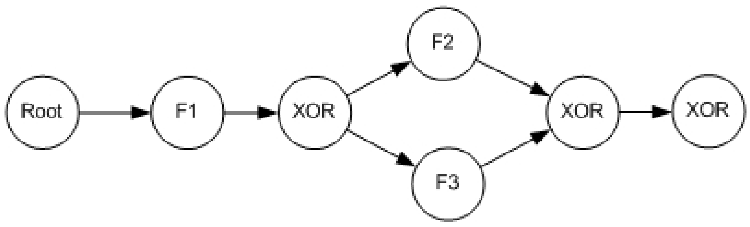


Figure 20: Simple Composition Example

we will ignore at this point the probability of each branch. This will be covered later in this section.

```
Global Constraints:
Response Time (rt)  <= 170ms,  required;
Availability (av) >= 85%, strong;

F1 Constraints:
Response Time (rt) <= 100 ms, required;
Availability (av) >= 95%, weak;

F2 Constraints:
Availability (av) >= 97%, weak;
```

Listing 30: List of Global and Local Constraints

A global required constraint is specified on response time and an optional on availability. Feature F1 contains also a required response time and an optional availability constraint. Feature F2 has only an optional availability constraint and F3 has no constraints. The following service candidates are given:

Feature name	F1			F2		F3	
Service Name	S1	S2	S3	S4	S5	S6	S7
Response Time	100ms	80ms	150ms	70ms	30ms	60ms	75ms
Availability	90%	97%	96%	96%	99%	93%	95%

Table 3: Service Candidates

Feature F1 has three service candidates, F2 and F3 have three service candidates each. The first and trivial task is to solve local (e.g., feature) required constraints. This is already done in the VRESCO query. When a list of service candidates are queried from VRESCO also all required local constraints are specified within to reduce and filter the number of results. This can be done because an unsatisfied local required constraint is equal to an over-constraint VCL composition that can't be created. In this example S3 is excluded already in this step.

At this time the following solutions are possible: (S1,S4,S6), (S1, S4, S7), (S1, S5, S6), (S1, S5, S7), (S2, S4, S6), (S2, S4, S7), (S2, S5, S6) ,(S2, S5, S7)

For global constraints an aggregation function needs to be specified for each QoS attribute. A set of features called in a sequence will sum their response times and multiply their availability. On a Xor split the maximum response time respectively the minimum availability of the two branches will be taken. A complete list of aggregation formulas for all QoS attributes will be given later in this section. The resulting aggregation formulas for this example will be:

$$global\,(rt) = S_{F1} + Max\,(S_{F2}, S_{F3})\,, S_{F1} \in F1, S_{F2} \in F2, S_{F3} \in F3$$

$$global\,(av) = S_{F1} * MIN\,(S_{F2}, S_{F3})\,, S_{F1} \in F1, S_{F2} \in F2, S_{F3} \in F3$$

Having both aggregation formulas, Table 4 gives an overview of the score and validity of each of the eight possible solutions.

Solution	**Global(rt)**	**Global(av)**	**F1(av)**	**F2(av)**	**Valid**	**Score**
(S1,S4,S6)	170	83,70%	90%	96%	YES	0
(S1,S4,S7)	170	85,50%	90%	96%	YES	10
(S1,S5,S6)	160	83,70%	90%	99%	YES	5
(S1,S5,S7)	175	85,50%	90%	99%	NO	15
(S2,S4,S6)	150	90,21%	97%	96%	YES	15
(S2,S4,S7)	155	92,15%	97%	96%	YES	15
(S2,S5,S6)	140	90,21%	97%	99%	YES	20
(S2,S5,S7)	155	92,15%	97%	99%	YES	20

Table 4: Optimization Result Table

The solution (S1,S5,S7) is the only invalid solution. All other solutions satisfied all required constraints. But considering optional constraints, global and local, there is a score from 0 to 20. The two best solutions are (S2,S5,S6) and (S2,S5,S6). Both solutions scores the maximum amount of points. (S2,S5,S6) has a slightly better global availability value. But this fact is ignored by the solver and it is possible to get the second solution as a result. The reason is that the solver tries to satisfies the constraints and not to take care of the QoS values itself. The decision that the first solution is better was easy in this case. But a composition can contain constraints for many different QoS values. In this case we can get two solution, one with a better response time, a second with a better availability and a third with a better cost value. The solver cannot make a decision which one to choose. They are all equal and the first one in the list will be returned. Having a large number of equal solutions is an indicator for under-constraint compositions. In this case, the user can specify stronger constraints.

After this simple example we can define the goal as an optimal selection of one service candidate $s_{ij} \in S_j$ for each feature f_j where all the required global and local constraints are satisfied and a number of optional constraints from the constraint hierarchy H are satisfied. We use two different approaches for modeling the QoS-aware optimization, a constraint optimization problem and an integer programming problem. The reason for devising an IP solution as an alternative is based on the fact that most constraint-based approaches have scalability problems when applied to medium and large-scale practical optimization problems, however, the COP solution provides a simpler way of handling constraint hierarchies. The definition of our CPS and IP problem was published in [59] and will be used unmodiefied to describe the COP and the IP approach more detailed. The following table shows all supported QoS attributes and their aggregation formulas.

Attribute	Unit	Sequence	Conditional (XOR)	Parallel (AND)	Loop
Response Time (q_{rt})	msec	$\sum_{i=1}^{n} q_{rt}(f_i)$	$\sum_{i=1}^{n} pi * q_{rt}(f_i)$	$max\{f_1,\ldots,f_n\}$	$q_{rt}(f) \times c$
Latency (q_l)	msec	$\sum_{i=1}^{n} q_l(f_i)$	$\sum_{i=1}^{n} pi * q_l(f_i)$	$max\{f_1,\ldots,f_n\}$	$q_l(f) \times c$
Price (q_p)	per invocation	$\sum_{i=1}^{n} q_p(f_i)$	$\sum_{i=1}^{n} pi * q_p(f_i)$	$\sum_{i=1}^{n} q_p(f_i)$	$q_p(f) \times c$
Availability (q_{av})	percent	$\prod_{i=1}^{n} q_{av}(f_i)$	$\sum_{i=1}^{n} pi * q_{av}(f_i)$	$\prod_{i=1}^{n} q_{av}(f_i)$	$q_{av}(f)^c$
Accuracy (q_{ac})	percent	$\prod_{i=1}^{n} q_{ac}(f_i)$	$\sum_{i=1}^{n} pi * q_{ac}(f_i)$	$\prod_{i=1}^{n} q_{ac}(f_i)$	$q_{ac}(f)^c$
Throughput (q_{tp})	percent	$min\{f_1,\ldots,f_n\}$	$\sum_{i=1}^{n} pi * q_{tp}(f_i)$	$min\{f_1,\ldots,f_n\}$	$q_{tp}(f)$
Reliable Messaging	$\{true, false\}$	$q'_{rm} = \begin{cases} true & \underset{1<i\leq n}{\forall} q_{rm}(f_i) = true \\ false & \underset{f_i \in F}{\exists} q_{rm}(f_i) = false \end{cases}$			
Security	$\{None, X.509, \ldots\}$	$q'_{sec} = \begin{cases} X.509 & \underset{1<i\leq n}{\forall} q_{seq}(f_i) = X.509 \\ None & otherwise \end{cases}$			

Table 5: QoS Attributes and Aggregation Formulas (from [59])

3.5.1 Constraint Optimization Problem

A COP is a constraint satisfaction problem (CSP) in which constraints are weighted and the goal is to find a solution maximizing a function of weighted constraints. A CSP is defined as a Tuple $\langle X, D, C \rangle$ where X represents a set of variables, D_i represents the domain of each variable X_i and C represents a set of constraints over the variable X. A solution is an assignment of values from the variable's domain D to each variable $X_i \in X$ satisfying all the constraints C. Both constraint types can have required and optional QoS constraints. Each required constraint has to be fulfilled, otherwise no solution can be found. All optional constraints (global and local) will be added to the objective function that has to be maximized. As aforementioned, all required feature constraint have already been pre-filtered, therefore, it is ensured that only service candidates have to be considered that fulfill all required feature constraints. This may reduce the number of constraints in the problem space.

a) Feature Constraints: Modeling feature constraints requires to add all service candidates S_j for each feature f_j as variables to the problem space. As we only want to select one service candidate from all available services S_j to execute a feature, we have to add the following selection constraint given that y_{ij} denotes the selection of a service candidate s_{ij} to execute a feature (y_{ij} is modeled as a boolean decision variable):

$$\sum_{i \in S_j} y_{ij} = 1, \forall j \in F$$

Each feature f_j can be subject to feature constraints, therefore, we need to add the following constraint for each feature constraint Q_{fc} to determine the selected QoS value q_{jk} of feature (local selection). q_{jk} represents the selected QoS value for a given feature f_j.

$$q_{jk} = \sum_{i \in S_j} Q_{ik} \times y_{ij}, \forall k \in Q_{fc}$$

Depending on the QoS attribute dimension(ascending, descending, exact) we need to add the corresponding constraints for each QoS attribute to capture whether an optional QoS constraint c_{jk} is satisfied (c_{jk} is represented as a Boolean decision variable). The value q_{jk} is the value from constraint given in the last formula. For descending dimension, $c_{jk} = (q_{jk} \leq Q_{fc_k})$ is added, for ascending dimension $c_{jk} = (q_{jk} \geq Q_{fc_k})$, and for exact dimension the resultim constraint is $c_{jk} = (q_{jk} = Q_{fc_k})$.

Additionally, we use the following function to map the constraint hierarchy levels to strength values that is then used to in the objective function. Please note that these values are flexible and can be changed to reflect a different mapping as discussed in previous section.

$$strength\,(c) = \begin{cases} 20 & if\ c \in H_1 \\ 10 & if\ c \in H_2 \\ 5 & if\ c \in H_3 \\ 0 & if\ c \in otherwise \end{cases}$$

All the aforementioned constraints describe the selection of an optional feature QoS value. These constraints are added for each feature f_j and maximized as part of the objective function:

$$max \sum_{j \in F} \sum_{k \in Q_{fc}} c_{jk} \times strength(Q_{fc_k})$$

b) Global Constraints : In order to add global constraints (required or optional ones), we first need to create an aggregation formula depending on the structured composition as shown in the example composition and the aggregation formulas shown in table 1. We use a recursive algorithm to traverse the structured composition from the previous step and generate an aggregation formula for each feature f_j. For example, when aggregation the response time for the composite block in Figure 1 (containing the $F1$ and $F2$ feature), the following aggregation constraint applies (k is the index for the QoS constraint, in this example it would be 0 for the response time):

$$ak = \max_{j \in \{F_2, F_3\}} \{q_{jk}\}$$

In the following, we use ak to represent the aggregation constraint of the k-th QoS attribute which is added for every global constraint that is specified by the user in the VCL script. In case the global QoS constraint is required, we add another constraint depending on the QoS attribute dimension. For QoS with descending dimension, $ak \leq Q_{gc_k}$ is added, for ascending dimension $ak \geq Q_{gc_k}$, and for exact dimension the resulting constraint is $ak = Q_{gc_k}$ represents the global QoS constraint where k is the QoS attribute index.

In case that global QoS constraint is optional, we have to add a decision constraint to check whether an optional constraint has been fulfilled. Again depending on the QoS attribute dimension, we add the following constraints: For descending dimension: $c_k = (ak \leq Q_{gc_k})$ is added, for ascending dimension $c_k = (ak \geq Q_{gc_k})$, and for exact dimension the resulting constraint is $c_k = (ak = Q_{gc_k})$. Finally, we have to add these decision constraints multiplied with their strength value to the objective function to get the overall objective function:

$$max \left(\sum_{j \in F} \sum_{k \in Q_{fc}} c_{jk} \times strength(Q_{fc_k}) + \sum_{k \in Q_{gc}} c_k \times strength(Q_{gc_k}) \right)$$

The objective funtion is then maximized by the solver to find an optimal solution within the constraint boundaries set by the user in teh VCL script. All the values in our COP are scaled to integers by multiplying them with 100. Due to the fact that we only allow two decimal places in VCL we do not have any precision loss.

3.5.2 Integer Programming Approach

An IP optimizes a liner objective function that is subject to linear equality an linear inequality constraints. Compared to the CSP approach, there are a few changes that are needed when modeling the QoS-aware composition problem as IP. We have to define a new objective function calculating an overall utility value for each feature fj considering the user's QoS constraints and their strength. Additionally, we need to linearize the aggregation rules for q_{ac} and a_{av} because they use the product to aggregate the QoS for a sequence and parallel execution of features.

a) Feature Constraints: Feature constraints are handled by using an utility function that is calculated for each service candidate. The selection constraint presented for the COP is still valid in the IP formulation. For calculating the QoS utility function for each service, we first need to scale all the QoS values to an uniform representation. Contrary to other approaches in this area, we do net use simple-additive weighting scaling the values, however, we scale all values between [0, 100] depending on the percentage to which a QoS attribute of service candidate fulfills the optional constraint imposed by the user. For example if the user specifies an optional availability constraint on a feature f_j with the value 0.95 and the QoS value if the service candidate is 0.99, we set the scaled value to 100 because the optional feature constraint is 100 percent satisfied (in fact is over-satisfied). The overall objective function is:

$$max \left(\sum_{j \in F} \sum_{i \in S_j} y_{ij} \times \sum_{k \in Q_{fc}} scale\left(Q_{ik}, Q_{fc_k}\right) \times strength\left(Q_{fc_k}\right) \right)$$

The function scale scales the k-th QoS value Q_{ik} of a service candidate si between [0,100] depending on the actual QoS feature constraint value Q_{fc_k} specified by the user in VCL and the QoS dimension (ascending, descending, exact). The COP strength formula is used.

b)Global Constraints: For adding the global constraint, we follow a similar approach as in the CSP solution. We first aggregate the QoS attribute using a similar function as in the CSP approach, with the exception that we linearize the product aggregation rules using the *ln*. Whenever a global QoS constraint is required, we add a linear equality or inequality to the problem space. If a global constraint is optional, we add it to the overall objective function that has to be maximized.

3.6 Composition Publishing

The last task in our approach is to publish the composition as a Web service. To simplify this task we use an existing composition framework and transform our structured composition to it. We choose Microsoft's Windows Workflow Foundation for this purpose. It is a workflow engine that is part of the .NET 3.0. Furthermore, WF has the ability to expose a workflow as a Web service within the Internet Information Server.

The first step is to transform the structured composition items to appropriate WF ActivityWorkflow items. Therefore, Table 1 shows the mapping between them.

Some items are natively supported and others are implemented within a code activity. WF comes with an InvokeWebService activity which can be used to call Web services. This is not used because we call VRESCO features and not native Web services. Therefore we use the DAIOS Framework, presented in the second chapter, in a code activity to call the selected service candidate. Furthermore, the `global`

Structured Item	**WF Activity Type**
Sequence	SequenceActivity
And Branch	ParallelActivity
Xor Branch (Check)	IfElseActivity
Whiel	WhileeActivity
Invoke	CodeActivity
Throw	CodeActivity
Return	CodeActivity

Table 6: WF Activity Mapping

feature, which is used as an entry point to the composition by having input and output parameters specified, is covered by a ReceiveActivity that is used to expose the composition with correct parameters and return values.

All code activities and the contract types for the used ReceiveActivity are defined in C# source code with CodeDom. Together with the generated XAML code it gets compiled to a dll that is published on the IIS and immediately available. The new WSDL endpoint address is returned to the user and the composition service call is finished.

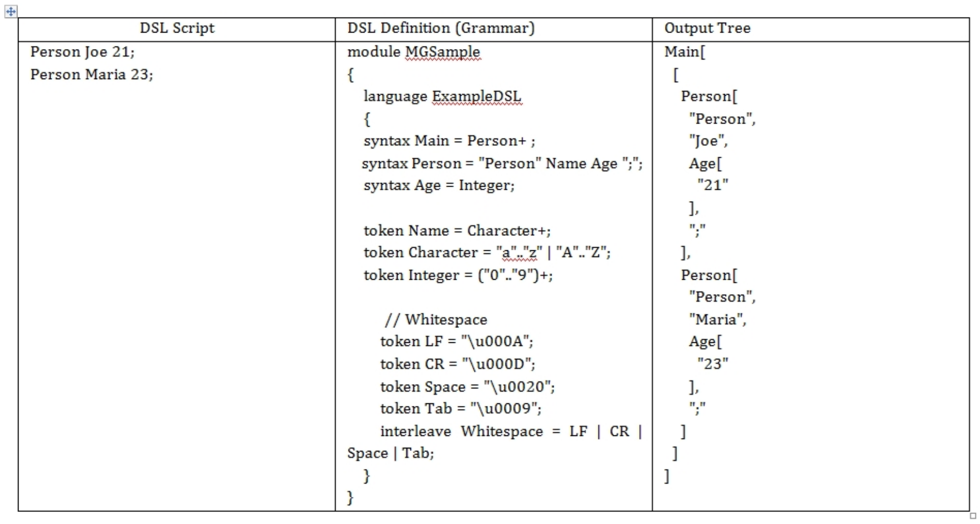
4 Implementation

This chapter will depict a number of implementation details of our approach. In the first section the MGrammar definition of VCL and the resulting semantic model will be shown. Section 2 covers the transformation from the semantic model to a structured composition. The required QoS optimization and solver implementation is part of section 3. The last section, section 4, deals with the transformation to a WF workflow and publishing to the IIS. We choose in this chapter only the core parts of the implementation to keep the size of this thesis in a reasonable range.

4.1 VLC Implementation

There are many ways to design and create a domain specific language. A lot of tools are offered for this purpose. At the beginning of this thesis we try to implement VCL by extending Ruby. This was, at that time, a popular way to create DSLs. Ruby offers a lot of possibilities to extend the language but we reach also the limits and have been forced to try a different way. At this time Microsoft announced the Oslo framework. Oslo give us the possibility to implement easily our desired syntax. We switch to MGrammar, a Oslo language for dsl syntax definition, and implement the VCL language. One of our goals was also not to create a lex/yacc solution and stay at the .NET platform, which is the case with MGrammar.

MGrammar specifications are purely declarative. They act like a rule set for a transformation engine. The resulting transformation is always a labeled tree and can be only slightly modified. Figure 21 is showing a simple transformation.



DSL Script

```
Person Joe 21;
Person Maria 23;
```

DSL Definition (Grammar)

```
module MGSample
{
   language ExampleDSL
   {
   syntax Main = Person+ ;
  syntax Person = "Person" Name Age ";";
   syntax Age = Integer;

   token Name = Character+;
   token Character = "a".."z" | "A".."Z";
   token Integer = ("0".."9")+;

      // Whitespace
     token LF = "\u000A";
     token CR = "\u000D";
     token Space = "\u0020";
     token Tab = "\u0009";
     interleave  Whitespace  =  LF  |  CR  |
Space | Tab;
   }
}
```

Output Tree

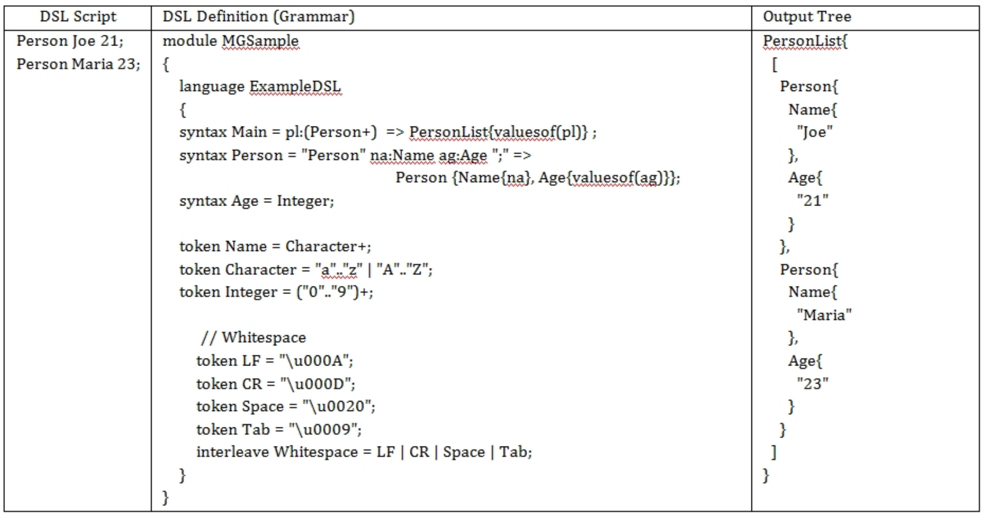
```
Main[
  [
   Person[
    "Person",
    "Joe",
    Age[
     "21"
    ],
    ";"
   ],
   Person[
    "Person",
    "Maria",
    Age[
     "23"
    ],
    ";"
   ]
  ]
]
```

Figure 21: MGrammar Example

A MGrammar definition consists of one or more named rules, each of them describes a part of the language. The specified language is named `ExampleDSL`. A language is built out of set of rules. The name Main is used as an initial rule that all input documents must match in order to be considered valid with respect to the language.

Rules use patterns to describe the set of input values that the rule applies to [47]. If we type `Person Joe 21;` the language processor will report that the input is valid. In Listing 21 the syntax and token rules are used. Syntax rules can be seen as higher level rules, and tokens more as a low level textual constructs in a language.

MGrammar transforms the DSL script to a tree structured. The shape and content of that tree is determined by syntax rules of the DSL definition. Each syntax rule consists of the rule definition itself and an optional projection. A projection describes how to transform the output. Figure 22 shows the same sample from above with a projection that is used to make the tree output more familiar for further processing and remove unnecessary elements like the semicolons.



DSL Script

```
Person Joe 21;
Person Maria 23;
```

DSL Definition (Grammar)

```
module MGSample
{
   language ExampleDSL
   {
   syntax Main = pl:(Person+)  => PersonList{valuesof(pl)} ;
   syntax Person = "Person" na:Name ag:Age ";" =>
                                  Person {Name{na}, Age{valuesof(ag)}};
   syntax Age = Integer;

   token Name = Character+;
   token Character = "a".."z" | "A".."Z";
   token Integer = ("0".."9")+;

      // Whitespace
     token LF = "\u000A";
     token CR = "\u000D";
     token Space = "\u0020";
     token Tab = "\u0009";
     interleave Whitespace = LF | CR | Space | Tab;
   }
}
```

Output Tree

```
PersonList{
 [
  Person{
   Name{
    "Joe"
   },
   Age{
    "21"
   }
  },
  Person{
   Name{
    "Maria"
   },
   Age{
    "23"
   }
  }
 ]
}
```

Figure 22: MGrammar Example with modified Output Tree

The MGrammar in Listing 22 removes all semicolons and provides a more common tree structure. Projections are used for the Main and Person rule. MGrammar has a lot more to offer that the described functionality. For more details the MGrammar Language Specification [47] published by Microsoft is a good reference. The VCL language is defined by using exactly the described basic principles. Appendix C contains the whole VCL MGrammar file.

A valid transformed DSL script to a tree structure needs to be forwarded in the composition chain. To make the tree useful it needs to be parsed to an object structure that can be used in C#. The MGrammar compiler provides a XAML output format that is used in in our approach. The resulting XAML code can be de-serialized to a given object mode. The target model for a VCL specification is depicted in Figure 23.

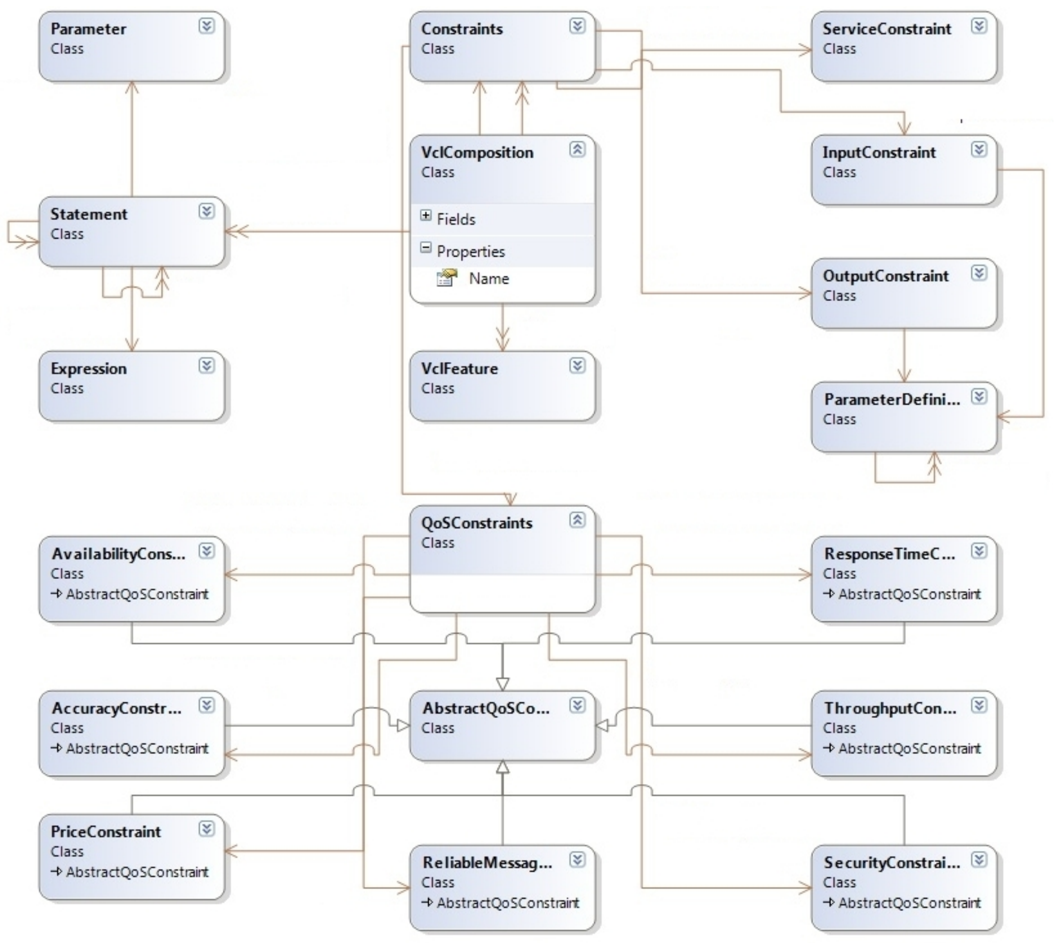


Figure 23: VCL Domain Model

The instance of the given model is used for further processing. The tranformation of the DSL script to the object model can be done on the client side with the client library or within the composition service that accepts both, a VCL specification or directly the object model over SOAP. Both approaches are supported to be able to create VCL clients without the usage of the client library. This allows an implementation in other languages and platforms than .NET.

The resulting object model is forwarded to dataflow resolving.

4.2 Creating a Structured Composition

With the provided object model from the previous step a structured composition can be created for further processing steps. In general the algorithm provided by

Eshuis et al. with the presented changes in the CaaS chapter is implemented.

4.2.1 Abstract Dependency Graph

The first step is to create an abstract dependency graph. Therefore, we extract for every feature its input and output values. An example is given in Table 7:

VCL	Implementation
`feature ActivatePort,`	
`*.PhoneManagementService.ActivatePortedNumber;`	
`constraint ActivatePort`	`List FeatureOutputs = new List();`
`{`	
	`FeatureOutputs.Add(`
`output = {`	`ActivatePort.ActivatePortedNumberResponse);`
`ActivatePortedNumberResponse[`	`FeatureOutputs.Add(`
`string Status;`	`ActivatePort.ActivatePortedNumberResponse.Status);`
`]`	
`}`	

Table 7: Extract Output Values for ADG

For every feature all nested data concepts are added to a global output list. The name of the feature is used as a prefix. The reason is to know when a matching is found to which feature it belongs. This means, an instance of a feature and not to the feature type is used because of the possibility to have more than one feature instances of the same underlying feature type in the VCL script.

The feature output list is filled iteratively. To be exact, the statements from the workflow list are iterated through. Every statement is interpreted by following this rules:

- If the statement is a simple invoke statement than the feature is extracted, a dependency to the current root node is added, the output data concept is extracted and the feature output list is searched for a match. If a dependency match is found a dependency between the features is added. Finally the features output data concepts are added to the feature list.
- If a complex statement is found (e.g. If-Then-Else) all dummy features are added and nested statements are interpreted recursively.

At the end the created ADG is iterated through to eliminate all violating dependencies. The final ADG as a list of features and a dictionary of dependencies is forwarded to the structured composition algorithm.

4.2.2 Structured Composition

The algorithm is an exact implementation of Eshuis proposed solution with an additional annotation of the XOR and AND blocks. Figure 24 depicts the resulting class hierarchy of the structured composition which allows a QoS optimization and the final transformation to a workflow engine program.

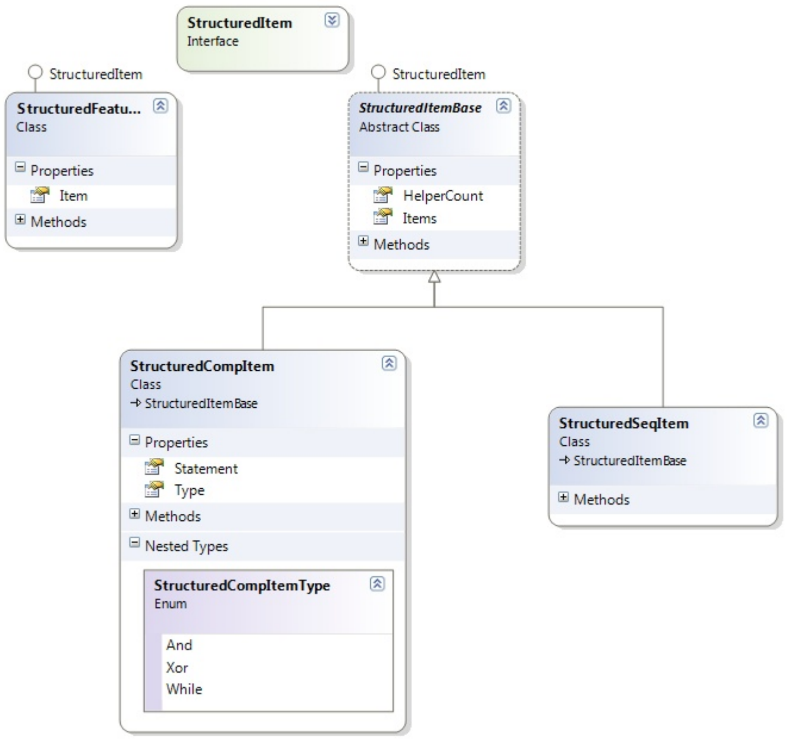


Figure 24: Structured Composition Domain Model

StructureItem is the base interface for all classes involved in the structured composition model. A COMP block is mapped to the StructuredCompItem class. An internal Enum is used to define the annotation types And, Xor or While. The base VCLStatement is also included to have internal details, like approximated while loops count or branching probabilities of an if-then-else statement, later available. For Seq blocks the StructuredSeqItem class is used. Both classes, StructuredSeqItem and StructuredCompItem, are derived from the abstract class StructuredItemBase. A basic feature is assigned to the StructuredFeatureItem.

The return value defined in the signature of the exposed algorithm method is StructuredItem. The interface includes also a set of methods which are necessary to access the inner data of the structured composition in later steps.

4.3 Quality of Service Optimization

Having a structured composition for the provided VCL script, the composition service performs a service candidate selection for each feature. The user specifies required and optional QoS constraints for the whole composition and single features that need to be considered in the selection process. The problem model was introduced in the CaaS chapter. In this section the implementation details will be covered. Therefore, source code parts will be used to explain our solution.

At the beginning of this thesis we started to model our problem for the Cassowary.net constraint solver. Cassowary.net is a port of the University of Washington's Cassowary constraint solving toolkit to the .NET platform. The definition of the solver provided on the Cassowary website is:

Cassowary is an incremental constraint solving toolkit that efficiently solves systems of linear equalities and inequalities. Constraints may be either requirements or preferences. Client code specifies the constraints to be maintained, and the solver updates the constrained variables to have values that satisfy the constraints.

Starting to model our problem with Cassowary we get problems to express it in the syntax provided by Cassowary. In general, it supports only basic equalities and inequalities. Every logical constraint requires a set of Cassowary constraints to be mapped correctly. In the end we decide to look for a different solver.

The next promising solver was NSolver developed and provided by Dr. Andy Chun from the Department on Computer Science at the City University of Hong Kong. The solver is constraint based and allows the user to specify a huge number of different operator within a constraint. With an easy to use syntax, we model our optimization problem without any troubles. After our first performance tests we were disappointed. It turns out that the solver has a great performance on a "flat" structured optimization problem, but has a performance issue with our model. We model our optional constraint with nested conditional statement provided by the NSolver. This has a huge impact on the performance.

At this time Microsoft announced their Microsoft Solver Foundation Framework for .NET. The framework itself is a container for different solver. A solver can be provided by Microsoft or a third-party company as an extension by implementing a set of predefined interfaces. Microsoft delivers the MSF Framework with a set of included solvers, among other a CSP and an IP solver.

We started to model our problem for the provided CSP solver in the *ConstraintSolverCSP* class. To have the possibility to support more than one implementation we define the *IConstraintSolver* interface that is implemented by the *ConstraintSolverCSP* class as well as the IP and the NSolver implementation.

The *ResolveConstraintHierarchie* method from Listing 31 shows the entry method for the Solver. It takes a Workflow instance as input which contains the structured

```
1  public void ResolveConstraintHierarchie(Core.Workflow wf)
2  {
3    SetGlobalConstraintFlags(wf);
4    AddFeatureParameters(wf.GraphItems);
5    AddGlobalConstraints(wf);
6    AddOverallScore();
7    Solve();
8
9    // cleanup stuff
10   _context.ClearModel();
11 }
```

Listing 31: ResolveConstraintHierarchie Method

composition . In Line 3 the *SetGlobalConstraintFlags* method is called to set a flag for each included global constraint that is used later in the implementation. All local constraints, required and optional, are handled in the *AddFeatureParameters* method (Line 4). Global constraints are covered with the *AddGlobalConstraints* call (Line 5). The utility function that needs to be maximized to get the best score on the optional constraints is generated in the *AddOverallScore* method (Line 6). Finally, having a setup for all constraints the Solve method is called to find a valid solution (Line 7). Line 10 cleans some MSF internal states to setup the solver ready for new calls.

```
private void SetGlobalConstraintFlags(VRESCo.Composition.Core.Workflow wf)
{
  if (wf.VclModel.GlobalConstraints != null)
  {
    var cons = wf.VclModel.GlobalConstraints.QoSConstraints;
    if (cons != null)
    {
      if (cons.AccuracyConstraint != null)
        hasGlobalAccuracyConstraint = true;
      if (cons.AvailabilityConstraint != null)
        hasGlobalAvailabilityConstraint = true;
      if (cons.PriceConstraint != null)
        hasGlobalPriceConstraint = true;
      if (cons.ReliableMessageConstraint != null)
        hasGlobalReliableMessageConstraint = true;
      if (cons.ResponseTimeConstraint != null)
        hasGlobalResponseTimeConstraint = true;
      if (cons.SecurityConstraint != null)
        hasGlobalSecurityConstraint = true;
      if (cons.ThroughputConstraint != null)
        hasGlobalThroughputConstraint = true;
    }
  }
}
```

Listing 32: SetGlobalConstraintFlags Method

The *SetGlobalConstraintFlags*(Listing 32) method extracts the global QoS constraints from the VCLModel (Line 5). It sets a global flag for each constraint that is given in the VCL file (Line 8-21). This is important for further processing and will be reused in other methods.

```
1  private void AddFeatureParameters(IList<AbstractDependencyGraphItem> items)
2  {
3    _candidateCount = 0;
4    foreach (var item in items)
5    {
6      if (item.Feature != null)
7      {
8        var featureName = item.Feature.Name;
9        if (item.ServiceCandidates.Count == 0)
10       {
11         throw new ConstraintResolutionException(string.Format("Cannot find
               a service candiates for features '{0}'.", featureName));
12       }
13
14       // create the global descision store for each feature
15       _featureDecisions[featureName] = new Dictionary<string, Decision>();
16
17       var candidates = item.ServiceCandidates;
18       _candidateCount += candidates.Count;
19
20       var services = new Set(Domain.Any, featureName);
21       var selected = new Decision(Domain.Boolean, featureName + "_Selection
           ", services);
22       _model.AddDecision(selected);
23
24       // add selection constraint (only one service can be selected for
             each feature)
25       _model.AddConstraint("SingleFeatureSelectionConstraint", Model.Sum(
             Model.ForEach(services, i => selected[i])) == 1);
```

Listing 33: AddFeatureParameters Method

AddFeatureParameter(Listing 33) is one of the main methods of the solver implementation. It adds the features and their local constraints to the solver model. In Line 2 it starts to iterate over the provided ADG Item list. The processing is started if the item is a feature (Line 4). This is an important check because the List contains also dummy nodes like Then or Throw items. The name of the feature is stored in the variable `featureName` (Line 5). If no service candidates are available for the feature an exception is thrown and the execution of the composition service is stopped. MSF provides Decision and Constraint objects that can be added to the solver to define a CSP Problem. The first sub-problem is to constraint that one and only one service candidate is selected for each feature. Therefore a set is defined and used in a Decision (Line 17-18). A Decision represents a value that needs to be assigned by the solver. Every Decision has a type, name and optionally a set. We use the defined set to indicate that we have an array of Boolean Decision, one for each service candidate. We add the `_Selection` string to the features name and

```
1  QoSConstraints qos = item.Constraint.QoSConstraints;
2  //////////
3  // ResponseTime local constraint
4  //////////
5  if ((qos != null && qos.ResponseTimeConstraint != null) ||
       hasGlobalResponseTimeConstraint)
6  {
7    var domain = Domain.IntegerNonnegative;
8    var responseTime = new Parameter(domain, Constants.QOS_RESPONSE_TIME,
         services);
9    _model.AddParameter(responseTime);
10   responseTime.SetBinding(candidates.AsEnumerable(), "QoSResponseTime", "
         Index");
11
12   if (qos != null && qos.ResponseTimeConstraint != null)
13   {
14     ResponseTimeConstraint rt = qos.ResponseTimeConstraint;
15     AddMinimiumFeatureConstraint(services, selected, responseTime, domain,
           rt.Value, rt.Strength, featureName, SELECTED_RESPONSE_TIME);
16   }
17   else
18   {
19     if (hasGlobalResponseTimeConstraint)
20     AddMinimiumFeatureConstraint(services, selected, responseTime, domain,
           0, Strength.None, featureName, SELECTED_RESPONSE_TIME);
21   }
22 }
```

Listing 34: AddFeatureParameters Method Continued

use it as a decision name. Line 19 adds finally the constraint by applying a `Sum` and `ForEach` operation on the decision to limit the sum to be 1. Considering that the MSF Boolean type has the value one for true and zero for false we express a constraint that only one Boolean within the array is true. The decision itself is added to the model(Line 20) and used later in the local and global constraints.

Listing 34 depicts the handling of the response time local constraint. Line 3 checks if an local or global response time constraint exists. In this case a new MSF Parameter is defined that holds the response time values for all service candidates (Line 5). The binding to response time values for service candidates is done by calling the *SetBinding* method (Line 6) and providing the list of service candidates as an enumerable, `QoSResponseTime` field for the response time value and `Index` filed for internal MSF indexing. The helper method *AddMinimumFeatureConstraint* is called to set the local constraint (Line 7 and Line 10). If an local constraint exist, the method will take the constraint value (e.g. `rt.Value`) as a parameter and constraint strength (e.g. `rt.Strength`), otherwise in case of a global constraint zero and `Strength.None` will be past as a parameters. All other QoS attributes are handled similar. The main difference is the helper method that is used. Therefore, we will not cover other QoS attributes here and will continue with the *AddMinimumFeatureConstraint* method in Listing 35.

```
1  private void AddMinimiumFeatureConstraint(Set services, Decision selected,
       Parameter parameter, Domain domain,
2   int qosValue, Strength strength, string featureName, string
        selectedDecisionName)
3  {
4    var selectedValue = new Decision(domain, selectedDecisionName);
5    _model.AddDecision(selectedValue);
6    _featureDecisions[featureName][selectedDecisionName] = selectedValue;
7    _model.AddConstraint(selectedDecisionName + "Constraint_" + featureName,
         selectedValue == Model.Sum(Model.ForEach(services, i => selected[i] *
          parameter[i])));
8
9    if (strength != Strength.Required && strength != Strength.None)
10   {
11     var constraintSelected = new Decision(Domain.Boolean, featureName +
           selectedDecisionName + "Optional");
12     _model.AddDecision(constraintSelected);
13
14     // add optional constraint and give a score according to its strength.
15     _model.AddConstraint("Optional" + selectedDecisionName + "_" +
           featureName, constraintSelected == (selectedValue <= qosValue));
16
17     _optionalConstraints.Add(constraintSelected * (int)strength);
18   }
19 }
```

Listing 35: AddMinimiumFeatureConstraint Method

In Line 3 a MSF Decision is created that will store the response time value of the selected service candidate. It is added to the model (Line 4) and to the `_featureDecisions` internal dictionary. The `_featureDecisions` directory holds for every feature and its QoS values a MSF Decision. This is used later for global constraint that are build out of them. Line 5 defines the decision value as a sum of selected and parameter set multiplications. This is exactly the defined formula introduced in the CaaS Chapter. The selected set is already constraint to have only one boolean variable with value true. Therefore, the decision variable will always have the response time value of the selected constraint.

In case of an optional constraint a decision variable, constraint to the value of the constraint strength if satisfied, will be added to the `_optionalConstraints` List. This is used later to define the score function that needs to me maximized.

The next step is to create global constraints. This is done in the AddGlobalConstraints method (Listing 36). Line 1 check if global QoS constraints are defined in the VCL specification. A MSF Decision is again created to hold the response time value of the whole composition(Line 5). The constraint used to define the value of the decision variable calls the *AggregateQosParameter* method. This method will not be covered in detail here because of its length. In general, it transverse the composition and concatenate the decisions from the `_featureDecisions` dictionary to a

```
private void AddGlobalConstraints(Core.Workflow wf)
{
  // global constraints defined in VCL?
  if (wf.VclModel.GlobalConstraints == null || wf.VclModel.
      GlobalConstraints.QoSConstraints == null)
    return;

  QoSConstraints global = wf.VclModel.GlobalConstraints.QoSConstraints;
  if (global != null && global.ResponseTimeConstraint != null)
  {
    // ResponseTime required
    var rt = global.ResponseTimeConstraint;

    _globalResponseTime = new Decision(Domain.IntegerNonnegative, "
        GlobalResponseTime");
    _model.AddDecision(_globalResponseTime);

    _model.AddConstraint("GlobalResponseTimeAggregationConstraint",
        _globalResponseTime == AggregateQoSParameter(wf.
        StructuredComposition, SELECTED_RESPONSE_TIME, AggregationType.Max,
          AggregationType.Max, AggregationType.Sum));

    if (rt.Strength == Strength.Required)
    {
      _model.AddConstraint("GlobalResponseTimeConstraint",
          _globalResponseTime <= rt.Value);
      //we don't need to maximize this because we are maximizing the score
          value.
      //_model.AddGoal("MinimizeGlobalResponseTime", GoalKind.Minimize,
          _globalResponseTime);
    }
    else // optional constraints
    {
      // store descisions globally
      var globalResponseTimeConstraint = new Decision(Domain.Boolean, "
          OptionalGlobalResponseTimeSelection");
      _model.AddDecision(globalResponseTimeConstraint);

      // add optional constraint and give a score according to its strength
          .
      _model.AddConstraint("OptionalGlobalResponseTimeSelectionConstraint",
           globalResponseTimeConstraint == (_globalResponseTime <= rt.Value
          ));

      _optionalConstraints.Add(globalResponseTimeConstraint * (int)rt.
          Strength);
    }
  }
```

Listing 36: AddGlobalConstraints Method

```
1  private void AddOverallScore()
2  {
3    //check if at least one optional constraint exists
4    if (_optionalConstraints.Count > 0)
5    {
6      var totalscore = new Decision(Domain.IntegerNonnegative, "TotalScore");
7      _model.AddDecision(totalscore);
8      _model.AddConstraint("TotalScoreConstraint", totalscore == Model.Sum(
           _optionalConstraints.ToArray()));
9      _model.AddGoal("TotalScore", GoalKind.Maximize, totalscore);
10   }
11 }
```

Listing 37: AddOverallScore Method

global constraint (Line 5). Thereby, the provider `AggregationType` parameters for `XOR`, `AND` and `SEQ` are used to perform the right aggregation operation. For response time the max function is used for both branch types and a sum for a sequence.

If the global constraint is required than a simple constraint is added to the model (Line 10), otherwise a decision variable is created and constraint to have the given strength value if satisfied. Line 15 adds the decision to the `_optionalConstraints` list to use it for the scoring function. All other QoS Attributes are handled in same way with other AggregationTypes. Listing 37 shows the *AddOverallScore* method. With all global and local constraints created in the previous methods the global scoring function can be created. This is done by summing up all optional constraint from the `_optionalConstraints` list. Line 5 adds a maximization goal for the model. The solver tries to maximize the given decision, which is equal to the goal to satisfied the maximum number of optional constraints weighted by constraints strength.

The Solve method will trigger the MSF solver (Listing 38). In this case we use MSFs CSP solver(Line 3). Line 4 start the solver. All found solutions are added to the `_solutions` list. When at least one solution is found the *AssignBestSolution* method will be called to assign the best service candidate, extracted from the first solution in the solutions list, to each feature in the structured composition.

We cover in this section the CSP implementation. In our approach we used also an IP model. Both solutions are similar. Therefore, we will not cover the IP model here. The main difference was already explained in the previous chapter.

```
internal void Solve()
{
  var cspDirective = new ConstraintProgrammingDirective()
  {
    TimeLimit = 60000 * 5 };

  MSSolution sol = _context.Solve(cspDirective);
  w.Stop();
  _solvingTime = w.ElapsedMilliseconds;

  Report report = null;

  if (sol != null && sol.Quality != SolverQuality.Infeasible && _solutions.
      Count < 1)
  {
    //add solution to global solutions list
    _solutions.Add(sol);
}
```

Listing 38: Solve Method

4.3.1 Windows Workflow Implementation

The created structured composition needs to be published on the IIS to finish the composition request. Following the rules presented in the mapping table in chapter 3 all items are mapped to appropriate WF items. Therefore, we transverse the structured composition and create the correct WF activity for each node. The resulting workflow is stored in a XAML file.

Unfortunately, the contend of the code activity can't be stored in a XAML file. Therefore, we use .NETs CodeDom to create the required source code on the fly. This is also the case for conditions within a while or if-then nodes and the root ReceiveActivity item. Listing 39 shows the generated source code for CRM request from the TELCO example.

We use the DAIOS framework to call the selected service candidate for a feature. The DAIOS framework is also used because of the possibility to implement dynamic rebinding to another service. The reason for a rebinding can be a QoS value change or a fallback if the service gets unavailable. A rebinding implementation is out of scope and left for further work. Line 3-12 set the Web service source for the DAIOS client and calls the service. `CRM_CustomerLookupRequest` and `CRM_CustomerLookupResponse` are generated from the VCL specification and used within the WF workflow to share the result from each call to the following feature calls. In the TELCO sample, the output data from the CRM feature is used as an input for the NotifyUser feature. The DAIOS Framework uses an own datastructure for input and output parameters that needs to be mapped. This is done in Line 5-8 for the input parameter and Line 12-20 for output parameters.

```
public virtual void Crm_ExecuteCode(object sender, EventArgs args)
{
  ServiceFrontendFactory factory = ServiceFrontendFactory.GetFactory("
      VReSCO.NDaios.NativeInvoker.NativeInvokerFactory");
  string url = "http://localhost:12000/CustomerService?wsdl";
  DaiosMessage message = new DaiosMessage();
  DaiosMessage val = new DaiosMessage();
  val.SetInt("CustomerId", this.customerId);
  message.SetComplex("request", val);
  ServiceFrontend frontend = factory.CreateFrontend(url);
  frontend.SetWSDLOperationName(new QName("CustomerLookup", null));
  frontend.SetEndpointAddress(url);
  DaiosMessage message3 = frontend.RequestResponse(message);
  this.Crm_CustomerLookupResponse = new CustomerLookupResponse();
  this.Crm_CustomerLookupResponse.Firstname = message3.GetComplex("
      CustomerLookupResult").GetString("Firstname");
  this.Crm_CustomerLookupResponse.Lastname = message3.GetComplex("
      CustomerLookupResult").GetString("Lastname");
  this.Crm_CustomerLookupResponse.PhoneNumber = message3.GetComplex("
      CustomerLookupResult").GetString("PhoneNumber");
  this.Crm_CustomerLookupResponse.Mail = message3.GetComplex("
      CustomerLookupResult").GetString("Mail");
  this.Crm_CustomerLookupResponse.Street = message3.GetComplex("
      CustomerLookupResult").GetString("Street");
  this.Crm_CustomerLookupResponse.Zip = message3.GetComplex("
      CustomerLookupResult").GetString("Zip");
  this.Crm_CustomerLookupResponse.City = message3.GetComplex("
      CustomerLookupResult").GetString("City");
}
```

Listing 39: CRM Feature Call Implementation

The two generated parts, Workflow XAML and the source code, are compiled together with the C# compiler into a dll file. This dll is moved to a newly created virtual directory on the IIS. The last step is to change the web.config template to fit to the new workflow and to return the new WSDL endpoint. The service is automatically available to the customer. Figure 25 depicts the final WF workflow of the TELCO example. A counter value is used for some WF activities to simplified the code generation and avoid conflicts. But there are no side effects to the produced WSDL file, which contains a clean interface description.

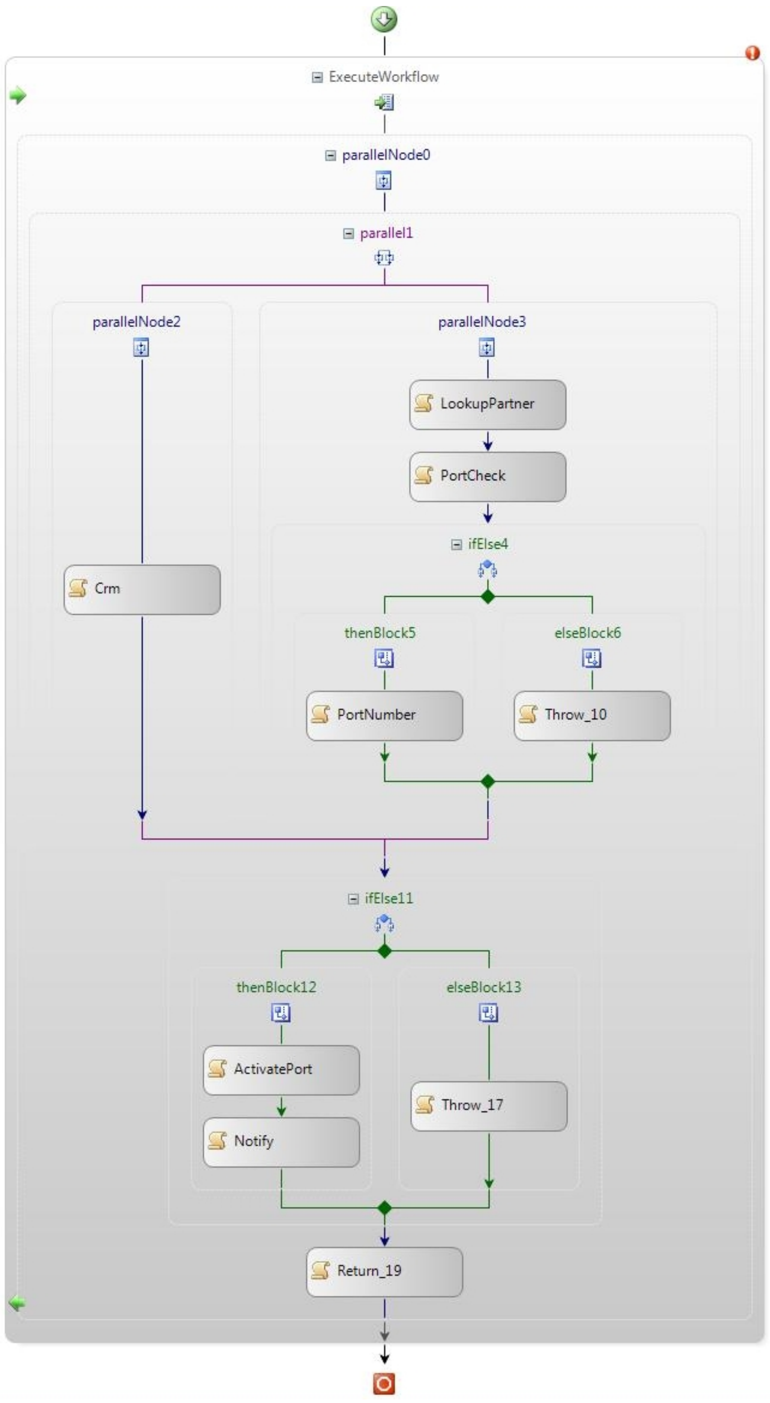


Figure 25: WF Workflow for TELCO Example

5 Conclusion and Future Work

Software as a Service is a software deployment and distribution model that grows in last year's. It becomes a valid business model to expose and license a specialized Web service on demand to customers. On the other side, companies can build software systems out of internal services or software components and use external services offers by third companies. Thereby, the development and maintaining cost can be highly reduced. Also the time to market can be decreased because the required services are already implemented and have a good quality.

Building complex software systems and using the SaaS approach creates the requirement for composition languages for Web services. BPEL is one of the wide spread languages that is used today.

This thesis introduces the service abstraction and QoS-aware composition language VCL (Vienna Composition Language). Services are abstracted to features within the VRESCO framework. This is used to map services with the same purpose but a different interface to an abstract domain service respectively feature in VRESCO terminology.

VCL defines the composition on a feature level. The workflow definition is not strictly defined like in BPEL. A combination of data-flow dependencies and control structures defined by the user are implemented. This allows rapid development of compositions in VCL.

Two services that maps to the same feature can differ in their QoS (e.g., Response time, Availability, Cost,...) values. Therefore, the best service candidate needs to be chosen. VCL supports local and global QoS constraints for this purpose. A list of implemented QoS values is given in Chapter 3. To avoid over-constraint compositions, each constraint can be enriched with a strength (e.g. required, hard, medium, weak). All non-required constraint are added to a scoring function that is maximized by the solver. Therefore, we model the optimization problem with a CSP and IP solver. CSP is easier to use and supports all constraint natively, whereas the IP implementation needs to use some approximations to implement the multiplication of QoS attributes exposed as percent values.

The resulting composition is transformed to WF and exposed as a Web service. We call this approach "Composition as a Service".

5.1 Future Work

However, there are some open points left for future work. In the current implementation the QoS constraint evaluation is done at the composition time. Typically, a QoS attribute like response time is permanently changing and needs to be measured. Therefore, the QoS evaluation needs to be moved to the composition runtime. The QoS optimization can be performed periodically and force the composition to rebind to new service candidates. Because of the fact, that the service call is done with the DAIOS client, which supports rebinding, this task should not be hard to implement. Another solution would be, instead of the periodically update, to monitor the QoS values of the deployed composition itself, an trigger a new QoS evaluation if the constraint are not satisfied.

The second topic is the QoS optimization itself. We implemented two solutions with the MSF framework. The resulting performance, at least for the IP model, was good enough for our approach. But if the VCL specification is big and the QoS optimization is moved to compositions runtime a better solver may be required. We think that there is enough space to optimize our current MSF models to get a better performance.

The VCL is closely coupled with the VRESCO framework. In our TELCO example we had to manually specify the data concepts used in each feature. This task should be automated by having a VCL Editor that can browse through VRESCOs registry and add features to the VCL per drag and drop. It is also possible to write a GUI editor or extend an existing developer IDE to model the complete composition. A good example is the WF editor in Visual Studio that offers an easy to use GUI for workflows and creates a XAML file as an output.

A Acronyms

BPEL Business Process Execution Language

BPML Business Process Modeling Language

CaaS Composition as a Service

CORBA Common Object Request Broker Architecture

CRM Customer Relationship Management

CSP Constraint satisfaction problem

CSS Cascading Style Sheets

DSL Domain-specific Language

GUI Graphical user interface

HQL Hibernate Query Language

HTML Hyper Text Markup Language

HTTP Hypertext Transfer Protocol

IIS Microsoft Internet Information Services

QoS Quality of Service

REST Representational State Transfer

RMI Remote Method Invocation

RPC Remote Procedure Call

SaaS Software as a Service

SMS Short Message Service

SOA Service-oriented architecture

SOAP Simple Object Access Protocol

UDDI Universal Description, Discovery and Integration

VCL Vienna Composition Language

VQL Vienna Querying Language

VRESCo Vienna Runtime Environment for Service-oriented Computing

WF Microsoft Windows Workflow Foundation

WSDL	Web Services Description Language
WSFL	Web Services Flow Language
XAML	Extensible Application Markup Language
XLANG	XML Language
XML	Extensible Markup Language

B VCL Specification for the TELCO Example

```
#compositions name
composition TelcoCasestudyComposition;

#*****************************
#** Define required features ***
#*****************************
feature Crm, *.CustomerService.CustomerLookup;
feature LookupPartner, *.PhoneManagementService.LookupPartner;
feature PortCheck, *.PortingService.PortabilityCheck;
feature PortNumber, *.PortingService.PortNumber;
feature ActivatePort, *.PhoneManagementService.ActivatePortedNumber;
feature Notify, *.NotificationService.NotifyUser;

#** Define feature constraints **

constraint global
{

 input = {
  int customerId;
  string numberToPort;
 }

 output = {
  string status;
 }
 qos = {
  responseTime = 7000;
  availability = 0.66;
 }

}

constraint Crm
{
 input = {
  CustomerLookupRequest[
   int CustomerId;
   ]
 }

 output = {
  CustomerLookupResponse[
   string Firstname;
   string Lastname;
   string PhoneNumber;
   string Mail;
   string Street;
   string Zip;
   string City;
  ]
 }

 qos = {
  responseTime = 1000;
  availability = 0.66;
 }
}

constraint LookupPartner
{
 input = {
  LookupPartnerRequest[
   string NumberToPort;
  ]
 }

 output = {
  LookupPartnerResponse[
   string ProviderName;
  ]
 }

 qos = {
  responseTime = 1000;
  availability = 0.66;
 }
}

```

```
constraint PortCheck
{
 input = {
  PortabilityCheckRequest[
   string NumberToPort;
   string NewProvider;
  ]
 }

 output = {
  PortabilityCheckResponse[
   int IsPortable;
  ]
 }

 qos = {
  responseTime = 1000;
  availability = 0.66,weak;
 }
}

constraint PortNumber
{
 input = {
  PortNumberRequest[
   string NumberToPort;
  ]
 }

 output = {
  PortNumberResponse[
   int IsPorted;
  ]
 }

 qos = {
  responseTime = 1000;
  availability = 0.66;
 }
}

constraint ActivatePort
{
 input = {
  ActivatePortedNumberRequest[
   int CustomerId;
   string PortedNumber;
  ]
 }

 output = {
  ActivatePortedNumberResponse[
   string Status;
  ]
 }

 qos = {
  responseTime = 1000;
  availability = 0.66;
 }
}

constraint Notify
{
 input = {
  NotifyRequest[
   string Mail;
   string Message;
   string ActivationStatus;
  ]
 }

 output = {
  NotifyResponse[
   string Status;
  ]
 }

 qos = {
  responseTime = 1000;
  availability = 0.66;
 }
}
```

```


#** Define activity ordering constraints (business protocol specification) *
# feature, array of parents

invoke Crm {
 CustomerLookupRequest[
  CustomerId = customerId;
 ]
}

invoke LookupPartner{
 LookupPartnerRequest[
  NumberToPort = numberToPort;
 ]
}

invoke PortCheck {
 PortabilityCheckRequest[
  NumberToPort = numberToPort;
  NewProvider = LookupPartner.LookupPartnerResponse.ProviderName;
 ]
}

check (PortCheck.PortabilityCheckResponse.IsPortable = 1)
{
 invoke PortNumber {
  PortNumberRequest[
   NumberToPort = numberToPort;
  ]
 }
}
else [0.1]
{
 throw "Number can't be ported by external provider";
}

check (PortNumber.PortNumberResponse.IsPorted = 1)
{
 invoke ActivatePort {
  ActivatePortedNumberRequest [
   CustomerId = customerId;
   PortedNumber = Crm.CustomerLookupResponse.PhoneNumber;
  ]
 }

invoke Notify {
 NotifyRequest [
  Mail = Crm.CustomerLookupResponse.PhoneNumber;
  Message = "Phone number ported";
  ActivationStatus = ActivatePort.ActivatePortedNumberResponse.Status;
  ]
 }
}
else [0.2]
{
 throw "Problem occurred on external partner side";
}

return {
 status = "Job done";
}
```

Listing 40: VCL Specification for the TELCO Example

C MGrammar Definition of VCL

```
module VRESCo
{
 language VCL
 {

  // Initial rule
  syntax Main = c:Composition
   f:FeatureDefinition+
   con:ConstraintDefinition*
   wf:(id:InvocationDefinition =>id|
   wd:JoinDefinition =>wd    |
   cd:CheckDefinition =>cd   |
   rd:ReturnDefinition =>rd  |
   td:ThrowDefinition =>td   |
   jd:JumpDefinition =>jd    |
   ld:LabelDefinition =>ld   |
   wd:WhileDefinition =>wd)*  => Composition {c, Features{f}, ConstraintCollection{con}, Workflow{wf}};

  // main rules
  syntax Composition = CompositionToken name:Identifier ";" => Name{name};
  syntax FeatureDefinition = FeatureToken locname:Identifier cat:(FullyQualifiedName) invT:(InvocationType)? ";" => Feature{Name{locname}, valuesof(cat), valuesof(invT)};
  syntax FullyQualifiedName = "," cp:FullyQualifiedNameToken => FullyQualifiedName{cp};
  syntax InvocationType = "," "type" inv:(Sync | Async) => InvocationType {valuesof(inv)};
  syntax Sync = SyncToken => "sync";
  syntax Async = AsyncToken => "async";

  syntax ConstraintDefinition = ConstraintToken id:Identifier "{"
   body:(it:QoSConstraint => it |
    it:InputConstraint => it |
    it:OutputConstraint => it |
    it:Precondition => it |
    it:Postcondition => it|
    it:ServiceConstraint =>it)* "}" => Constraints{FeatureName{id},valuesof(body)};

  syntax QoSConstraint = QoSToken "=" "{" qc:(
   it:QoSResponseTimeConstraint => it|
   it:QoSAvailabilityConstraint => it|
   it:QoSReliableMessageConstraint => it|
   it:QoSSecurityConstraint => it|
   it:QoSAccuracyConstraint => it|
   it:QoSThroughputConstraint => it|
   it:QoSPriceConstraint => it)* "}" => QoSConstraints{QoSConstraints{valuesof(qc)}};

  syntax QoSResponseTimeConstraint = ResponseTimeToken "=" st:( im:(qv:Digits ";" => [Value{qv}]) => im | iz:(qv:Digits "," li:Strength ";" => [Value{qv},Strength{li}]) =>iz ) => ResponseTimeConstraint{ResponseTimeConstraint{valuesof(st)}};
  syntax QoSAvailabilityConstraint = AvailabilityToken "=" st:( im:(qv:DoubleDigit ";" => [Value{qv}]) => im | iz:(qv:DoubleDigit "," li:Strength ";" => [Value{qv},Strength{li}]) =>iz ) => AvailabilityConstraint{AvailabilityConstraint{valuesof(st)}};
  syntax QoSAccuracyConstraint = AccuracyToken "=" st:( im:(qv:DoubleDigit ";" => [Value{qv}]) => im | iz:(qv:DoubleDigit "," li:Strength ";" => [Value{qv},Strength{li}]) =>iz ) => AccuracyConstraint{AccuracyConstraint{valuesof(st)}};
  syntax QoSThroughputConstraint = ThroughputToken "=" st:( im:(qv:DoubleDigit ";" => [Value{qv}]) => im | iz:(qv:DoubleDigit "," li:Strength ";" => [Value{qv},Strength{li}]) =>iz ) => ThroughputConstraint{ThroughputConstraint{valuesof(st)}};
  syntax QoSPriceConstraint = PriceToken "=" st:( im:(qv:DoubleDigit ";" => [Value{qv}]) => im | iz:(qv:DoubleDigit "," li:Strength ";" => [Value{qv},Strength{li}]) =>iz ) => PriceConstraint{PriceConstraint{valuesof(st)}};
  syntax QoSReliableMessageConstraint = ReliableToken "="  st:( im:(qv:Bool ";" => [Value{qv}]) => im | iz:(qv:Bool "," li:Strength ";" => [Value{qv},Strength{li}]) =>iz ) => ReliableMessageConstraint{ReliableMessageConstraint{valuesof(st)}};
  syntax QoSSecurityConstraint = SecurityToken "="  st:( im:(qv:( "X509" | "None" | "IntegratedSecurity" | "UsernamePassword") ";" => [Value{valuesof(qv)}]) => im |
   iz:(qv:("X509" | "None" | "IntegratedSecurity" | "UsernamePassword") "," li:Strength ";" => [Value{valuesof(qv)},Strength{li}]) =>iz ) => SecurityConstraint{SecurityConstraint{valuesof(st)}};

  syntax InputConstraint = InputToken "=" "{" it:ParameterDefinition "}"=> InputConstraint{InputConstraint{it}};
  syntax OutputConstraint = OutputToken "=" "{" it:ParameterDefinition "}" => OutputConstraint{OutputConstraint{it}};
  syntax Precondition = PrecondToken "=" "{" it:Parameter "}" => Precondition{Precondition{it}};
  syntax Postcondition = PostcondToken "=" "{" it:Parameter "}" => Postcondition{Postcondition{it}};
  syntax ServiceConstraint = ServiceToken "=" "{" "name" "=" it:Identifier ";" "}" => ServiceConstraint{ServiceConstraint{Name{it}}};
```

```

    syntax ParameterDefinition = it:ParameterDefinitionElement  => ParameterDefinition{ParameterDefinition{Type{"Complex"},Members{it}}} | id:Identifier "[" it:ParameterDefinitionElement "]" => ParameterDefinition{ParameterDefinition{Name{id},Type{"Complex"},Members{it}}} ;
    syntax ParameterDefinitionElement  = si:SimpleParameterDefinition => si | ci:ComplexParameterDefinition => ci;
    syntax SimpleParameterDefinition = t:Types name:Identifier ";" rest:ParameterDefinitionElement? => [ParameterDefinition{Type{t}, Name{name}}, valuesof(rest)] ;
    syntax ComplexParameterDefinition =  id:Identifier "[" it:ParameterDefinitionElement "]" ";" rest:ParameterDefinitionElement? => [ParameterDefinition{Name{id},Type{"Complex"},Members{it}}, valuesof(rest)] ;

    syntax Parameter = it:ParameterElement  => Parameter{Parameter{Type{"Complex"},Members{it}}} | id:Identifier "[" it:ParameterElement "]" => Parameter{Parameter{Name{id},Type{"Complex"},Members{it}}} ;
    syntax ParameterElement  = si:SimpleParameter => si | ci:ComplexParameter => ci;
    syntax SimpleParameter = name:Identifier "=" t:(Identifier | IdentifierWithDot | Text) ";" rest:ParameterElement? => [Parameter{Type{"Simple"}, Value{valuesof(t)}, Name{name}}, valuesof(rest)] ;
    syntax ComplexParameter =  id:Identifier "=" "[" it:ParameterElement "]" ";" rest:ParameterElement? => [Parameter{Name{id},Type{"Complex"},Members{it}}, valuesof(rest)] ;

    syntax InvocationDefinition = InvokeToken name:Identifier "{" p:Parameter  "}" => Statement{Type{"Invocation"},Feature{name},p};

    syntax JoinDefinition = JoinToken id:Identifier  rest:("," id:Identifier => Feature{Name{id}})* => Statement{Type{"Join"},Features{[Feature{Name{id}}, valuesof(rest)]}};

    syntax CheckDefinition = CheckToken "(" ex:Expression ")" "{" st:Statement "}" est:(ElseToken prob:("[" probinn:ProbabilityValue "]" => ElseBlockProbability{probinn})? "{" st:Statement "}" => {prob, StatementBlockElse{valuesof(st)}})? => Statement{Type{"Check"},Expression{valuesof(ex)},st, valuesof(est) };

    syntax WhileDefinition = WhileToken times:("[" timesinner:Digits "]"=>WhileTimes{timesinner})? "(" ex:Expression ")" "{" st:Statement "}" => Statement{times,Type{"While"},Expression{valuesof(ex)},st};

    syntax Statement = st:(wd:JoinDefinition =>wd | cd:CheckDefinition => cd | id:InvocationDefinition => id| td:ThrowDefinition =>td | rd:ReturnDefinition => rd| jd:JumpDefinition=>jd | ld:LabelDefinition => ld |wd:WhileDefinition => wd)* => StatementBlock{[valuesof(st)]};

    syntax Expression = exout:("(" ex:Expression ")"=> ex) =>exout | g:(lv:(Identifier | Digit* | Text | IdentifierWithDot) op:("=" | "!=" | "<" | ">") rv:(IdentifierWithDot | Digit* | Text | Identifier) => {LeftValue{valuesof(lv)}, Operator{valuesof(op)},RightValue{valuesof(rv)}})  =>{Expression{Type{"Simple"}, valuesof(g)}} | (exl:Expression left(1) "&"  exr:Expression => Expression{Type{"And"}, Left{valuesof(exl)} ,Right{valuesof(exr)}}) | (exl:Expression left(2) "|" exr:Expression => Expression{Type{"Or"}, Left{valuesof(exl)} ,Right{valuesof(exr)}});

    syntax ThrowDefinition = ThrowToken tx:Text ";" =>Statement{Type{"Throw"},Message{tx}};
    syntax ReturnDefinition = ReturnToken "{" p:Parameter  "}" => Statement{Type{"Return"},p};
    syntax JumpDefinition = JumpToken id:Identifier ";" =>Statement{Type{"Jump"},Label{id}};
    syntax LabelDefinition = LabelToken id:Identifier ";" =>Statement{Type{"Label"},Name{id}};

    // tokens
    token ProbabilityValue = "0."("1".."9");
    token Digit = ("0".."9");
    token Digits = Digit+;
    token DoubleDigit = Digits ("." Digits)?;
    token Bool = ("true" | "false");
    token Character = "a".."z" | "A".."Z" | "_" | "-";
    token Types = ("string" | "int" | "long" | "double" | "boolean" );
    token Text = "\"" (^"\"")* "\"";
    token Identifier = Character (Character|Digit)*;
    token IdentifierWithDot = Identifier ("." Identifier)+;
    token FullyQualifiedNameToken = ("*." )? (Identifier | IdentifierWithDot) ;

    @{Classification["Keyword"]}
    token Strength = "required" | "strong" | "medium" | "weak";

    @{Classification["Comment"]}
    token Comment = "#" ^("\r" | "\n")+;

    // Whitespace
    syntax LF = "\u000A";
    syntax CR = "\u000D";
    syntax Space = "\u0020";
    syntax Tab = "\u0009";
    interleave Whitespace = LF | CR | Space | Tab |Comment;

    @{Classification["Keyword"]}
    token AccuracyToken = "accuracy";
    @{Classification["Keyword"]}
    token ThroughputToken = "throughput";
    @{Classification["Keyword"]}
    token PriceToken = "price";
    @{Classification["Keyword"]}
    token FeatureToken = "feature";
    @{Classification["Keyword"]}
    token AsyncToken = "async";
    @{Classification["Keyword"]}
    token SyncToken = "sync";
    @{Classification["Keyword"]}
    token ThrowToken = "throw";
    @{Classification["Keyword"]}
    token InvokeToken = "invoke";
    @{Classification["Keyword"]}
    token LabelToken = "label";
    @{Classification["Keyword"]}
    token ReturnToken = "return";
    @{Classification["Keyword"]}
    token JumpToken = "jump";
    @{Classification["Keyword"]}
    token CompositionToken = "composition";
    @{Classification["Keyword"]}
    token CheckToken = "check";
    @{Classification["Keyword"]}
    token ElseToken = "else";
    @{Classification["Keyword"]}
    token JoinToken = "join";
    @{Classification["Keyword"]}
    token ConstraintToken = "constraint";
    @{Classification["Keyword"]}
    token QoSToken = "qos";
    @{Classification["Keyword"]}
    token ResponseTimeToken = "responseTime";
    @{Classification["Keyword"]}
    token AvailabilityToken = "availability";
    @{Classification["Keyword"]}
    token ReliableToken = "reliableMessage";
    @{Classification["Keyword"]}
    token SecurityToken = "security";
    @{Classification["Keyword"]}
    token InputToken = "input";
    @{Classification["Keyword"]}
    token ServiceToken = "service";
    @{Classification["Keyword"]}
    token OutputToken = "output";
    @{Classification["Keyword"]}
    token PrecondToken = "precond";
    @{Classification["Keyword"]}
    token PostcondToken = "postcond";
    @{Classification["Keyword"]}
    token WhileToken = "while";
  }
}
```

Listing 41: MGrammar Definition of VCL

References

[1] Greg J. Badros, Alan Borning, and Peter Stuckey. The Cassowary Linear Arithmetic Constraint Solving Algorithm. Technical report, ACM Transactions on Computer Human Interaction, 1998.

[2] Amit Bahree, Shawn Cicoria, Dennis Mulder, Nishith Pathak, and Chris Peiris. *Pro WCF: Practical Microsoft SOA Implementation.* Apress, 2007.

[3] Roman Barták. Algorithms for Solving Constraint Hierarchies. `http://ktiml.mff.cuni.cz/~bartak/constraints/ch_solvers.html`, 1998. [Online; accessed 01-November-2009].

[4] Jeremy Bolie, Michael Cardella, Stany Blanvalet, Matjaz Juric, Sean Carey, Praveen Chandran, Yves Coene, and Kevin Geminiuc. *BPEL Cookbook: Best Practices for SOA-based integration and composite applications development: Ten practical real-world case studies combining business ... management and web services orchestration.* Packt Publishing, 2006.

[5] David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris, and David Orchard. Web Services Architecture. `http://www.w3.org/TR/ws-arch/`, 2004. [Online; accessed 01-November-2009].

[6] David Booth and Canyang Kevin Liu. Web Services Description Language (WSDL) Version 2.0 Part 0: Primer. `http://www.w3.org/TR/2007/REC-wsdl20-primer-20070626`, 2007. [Online; accessed 01-November-2009].

[7] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, François Yergeau, and John Cowan. Extensible Markup Language (XML) 1.1 (Second Edition). `http://www.w3.org/TR/2006/REC-xml11-20060816/`, 2006. [Online; accessed 01-November-2009].

[8] Bruce Bukovics. *Pro WF: Windows Workflow in .NET 3.5.* Apress, 2008.

[9] Gerardo Canfora, Massimiliano Di Penta, Raffaele Esposito, and Maria Luisa Villani. A Framework for QoS-Aware Binding and Re-Binding of Composite Web Services. *J. Syst. Softw.*, 81(10):1754–1769, 2008.

[10] F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Workflow Evolution. In *Data and Knowledge Engineering*, pages 438–455. Springer Verlag, 1996.

[11] David Chappell. Introducing Windows Workflow Foundation. `http://msdn.microsoft.com/de-de/library/ee210343\%28en-us\%29.aspx`, 2009. [Online; accessed 01-November-2009].

[12] Roberto Chinnici, Hugo Haas, Amelia A. Lewis, Jean-Jacques Moreau, David Orchard, and Sanjiva Weerawarana. Web Services Description Language

(WSDL) Version 2.0 Part 2: Adjuncts. `http://www.w3.org/TR/2007/REC-wsdl20-adjuncts-20070626/`, 2007. [Online; accessed 01-November-2009].

[13] Roberto Chinnici, Jean-Jacques Moreau, Arthur Ryman, and Sanjiva Weerawarana. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. `http://www.w3.org/TR/2007/REC-wsdl20-20070626/`, 2007. [Online; accessed 01-November-2009].

[14] Luc Clement, Andrew Hately, Claus von Riegen, and Tony Rogers. Universal Description, Discovery and Integration v3.0.2 (UDDI). `http://www.oasis-open.org/committees/uddi-spec/doc/spec/v3/uddi-v3.0.2-20041019.htm`, 2005. [Online; accessed 01-November-2009].

[15] Paul Dourish, Jim Holmes, Allan Maclean, Pernille Marqvardsen, and Alex Zbyslaw. Freeflow: Mediating Between Representation and Action in Workflow Systems. pages 190–198. ACM Press, 1996.

[16] Schahram Dustdar, Harald Gall, and Manfred Hauswirth. *Software-Architekturen für Verteilte Systeme: Prinzipien, Bausteine und Standardarchitekturen für moderne Software (Xpert.press) (German Edition)*. Springer, 2003.

[17] Schahram Dustdar and Wolfgang Schreiner. A Survey on Web Services Composition. *Int. J. Web Grid Serv.*, 1(1):1–30, 2005.

[18] ECMA. Standard ECMA- 334 C# Language Specification. `http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-334.pdf`, 2006. [Online; accessed 01-December-2009].

[19] Rik Eshuis, Paul W. P. J. Grefen, and Sven Till. Structured Service Composition. In Schahram Dustdar, José Luiz Fiadeiro, and Amit P. Sheth, editors, *Business Process Management*, volume 4102 of *Lecture Notes in Computer Science*, pages 97–112. Springer, 2006.

[20] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures.* Dissertation for the doctor of philosophy in information and computer science, University of Califonia, Irvine, 2000. `http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf`.

[21] Martin Fowler. Domain Specific Language. `http://www.martinfowler.com/bliki/DomainSpecificLanguage.html`. [Online; accessed 01-November-2009].

[22] Martin Fowler. A Language Workbench in Action - MPS. `http://martinfowler.com/articles/mpsAgree.html`, 2005. [Online; accessed 01-November-2009].

[23] Martin Fowler. Generating Code for DSLs. http://martinfowler.com/articles/codeGenDsl.html, 2005. [Online; accessed 01-November-2009].

[24] Martin Fowler. Language Workbenches: The Killer-App for Domain Specific Languages? http://martinfowler.com/articles/languageWorkbench.html, 2005. [Online; accessed 01-November-2009].

[25] Gartner. Enterprise Applications-Adoption of E-Business and Document Technologies. http://www.aiim.org, 2000. [Online; accessed 01-December-2009].

[26] Diimitrios Georgakopoulos, Mark Hornick, and Amit Sheth. An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. In *DISTRIBUTED AND PARALLEL DATABASES*, pages 119–153, 1995.

[27] Danny M. Groenewegen, Zef Hemel, Lennart C.L. Kats, and Eelco Visser. WebDSL: A Domain-Specific Language for Dynamic Web Applications. In *OOPSLA Companion '08: Companion to the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications*, pages 779–780, New York, NY, USA, 2008. ACM.

[28] William Grosso. *Java RMI.* O'Reilly Media, 2001.

[29] Network Working Group. Hypertext Transfer Protocol – HTTP/1.1. http://www.ietf.org/rfc/rfc2616.txt, 1999. [Online; accessed 01-November-2009].

[30] Object Managemet Group. CORBA Component Model Specification. http://www.omg.org/spec/CCM/4.0/PDF/, 2006. [Online; accessed 01-December-2009].

[31] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen, Anish Karmarkar, and Yves Lafon. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). http://www.w3.org/TR/2007/REC-soap12-part1-20070427/, 2007. [Online; accessed 01-November-2009].

[32] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen, Anish Karmarkar, and Yves Lafon. SOAP Version 1.2 Part 2: Adjuncts (Second Edition). http://www.w3.org/TR/2007/REC-soap12-part2-20070427/, 2007. [Online; accessed 01-November-2009].

[33] Martin Gudgin, Marc Hadley, Tony Rogers, and Ümit Yalçinalp. Web Services Addressing 1.0 - Metadata. http://www.w3.org/TR/2007/REC-ws-addr-metadata-20070904/, 2007. [Online; accessed 01-November-2009].

[34] Diane Jordan and John Evdemon. Web Services Business Process Execution Language v2.0. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf, 2007. [Online; accessed 01-November-2009].

[35] Aaron Junod, Robert Bazinet, and Dan Bernier. *Professional IronRuby.* Wrox, 2009.

[36] David Langworthy, Brad Lovering, and Don Box. *The Oslo Modeling Language: Draft Specification - October 2008.* Addison-Wesley Professional, 2008.

[37] Philipp Leitner, Florian Rosenberg, and Schahram Dustdar. Daios: Efficient Dynamic Web Service Invocation. *IEEE Internet Computing*, 13(3):72–80, 2009.

[38] Phillip Leitner. The Daios Framework - Dynamic, Asynchronous and Message-oriented Invocation of Web Services, 2008.

[39] Frank Leymann. Web Service Flow Language. `http://xml.coverpages.org/WSFL-Guide-200110.pdf`, 2001. [Online; accessed 01-December-2009].

[40] C. Matthew, Ken Laskey, Francis McCabe, Peter F Brown, and Rebekah Metz. Reference Model for Service Oriented Architecture v1.0. `http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf`, 2006. [Online; accessed 01-November-2009].

[41] Marjan Mernik, Uros Novak, Enis Avdicausevic, Mitja Lenic, Viljern Zurner, and Viljem Zumer. Design and Implementation of Simple Object Description Language. In *In ACM Symposium on Applied Computing, SAC'2001*, pages 590–594, 2001.

[42] Marjan Mernik and Viljem Zumer. Domain-Specific Languages for Software Engineering - Minitrack Introduction. In *HICSS*, 2001.

[43] Marjan Mernik, Viljem Zumer, and Marjan Mernik Viljem. Reusability of Formal Specifications in Programming Language Description, 1997.

[44] Anton Michlmayr, Florian Rosenberg, and Schahram Dustdar. End-to-End Versioning Support for Web Services. In *In Proceedings of the International Conference on Services Computing (SCC 2008). IEEE Computer Society.* IEEE Computer Society, 2008.

[45] Anton Michlmayr, Florian Rosenberg, Philipp Leitner, and Schahram Dustdar. Advanced Event Processing and Notifications in Service Runtime Environments. In *DEBS '08: Proceedings of the second international conference on Distributed event-based systems*, pages 115–125, New York, NY, USA, 2008. ACM.

[46] 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.

[47] Microsoft. The "Oslo" Modeling Language Specification. `http://download.microsoft.com/download/D/A/B/DAB9E2D8-3A27-4BA7-BE66-8600EE4E33B0/M_Language_Specification.pdf`, 2006. [Online; accessed 01-November-2009].

[48] Microsoft. Microsoft Solver Foundation - Overview, 2009. Version 1.2.

[49] Microsoft. Microsoft Solver Foundation - SFS Parameter Binding Overview, 2009. Version 1.2.

[50] Microsoft. Microsoft Solver Foundation - Solver Foundation Services, 2009. Version 1.2.

[51] Microsoft. Microsoft Solver Foundation - Solver Programming Primer, 2009. Version 1.2.

[52] Christian Nagel, Bill Evjen, Jay Glynn, Morgan Skinner, and Karli Watson. *Professional C# 2008 (Wrox Professional Guides)*. Wrox, 2008.

[53] Ericsson Nilo Mitra. SOAP Version 1.2 Part 0: Primer (Second Edition). `http://www.w3.org/TR/2007/REC-soap12-part0-20070427/`, 2007. [Online; accessed 01-November-2009].

[54] Ernst Oberortner, Uwe Zdun, and Schahram Dustdar. Domain-Specific Languages for Service-Oriented Architectures: An Explorative Study. In *ServiceWave '08: Proceedings of the 1st European Conference on Towards a Service-Based Internet*, pages 159–170, Berlin, Heidelberg, 2008. Springer-Verlag.

[55] Cesare Pautasso and Gustavo Alonso. JOpera: A Toolkit for Efficient Visual Composition of Web Services. *Int. J. Electron. Commerce*, 9(2):107–141, 04-5.

[56] M. Pesic. DecSerFlow: Towards a Truly Declarative Service Flow Language. pages 1–23. Springer, 2006.

[57] M. Pesic, M. H. Schonenberg, and N. Sidorova. Constraint-Based Workflow Models: Change Made Easy. In *In CoopIS*, 2007.

[58] Ingo Rammer and Mario Szpuszta. *Advanced .NET Remoting, Second Edition.* Apress, 2005.

[59] Florian Rosenberg, Predrag Celikovic, Anton Michlmayr, Philipp Leitner, and Schahram Dustdar. An End-to-End Approach for QoS-Aware Service Composition. *Enterprise Distributed Object Computing Conference, IEEE International*, 0:151–160, 2009.

[60] Florian Rosenberg, Philipp Leitner, and Anton Michlmayr. A Metamodel for Enriching the Semantics of Service-centric Applications.

[61] Florian Rosenberg, Philipp Leitner, Anton Michlmayr, Predrag Celikovic, and Schahram Dustdar. Towards Composition as a Service - A Quality of Service Driven Approach. *Data Engineering, International Conference on*, 0:1733–1740, 2009.

[62] Florian Rosenberg, Philipp Leitner, Anton Michlmayr, and Schahram Dustdar. Integrated Metadata Support for Web Service Runtimes. *Enterprise Distributed Object Computing Workshops, International Conference on*, 0:361–368, 2008.

[63] Poornachandra Sarang, Matjaz Juric, and Benny Mathew. *Business Process Execution Language for Web Services BPEL and BPEL4WS 2nd Edition*. Packt Publishing, 2006.

[64] R. Srinivasan. RFC 1831: RPC: Remote Procedure Call Protocol Specification Version 2, August 1995. Status: PROPOSED STANDARD.

[65] Biplav Srivastava and Jana Koehler. Web Service Composition - Current Solutions and Open Problems. In *In: ICAPS 2003 Workshop on Planning for Web Services*, pages 28–35, 2003.

[66] Satish Thatte. XLANG Web Services for Business Process Design. `http://xml.coverpages.org/XLANG-C-200106.html`, 2006. [Online; accessed 01-December-2009].

[67] W. M. P. van der Aalst, Ter A. H. M. Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.

[68] Wil M.P. van der Aalst, Marlon Dumas, and Arthur H.M. ter Hofstede. Web Service Composition Languages: Old Wine in New Bottles? *EUROMICRO Conference*, 0:298, 2003.

[69] W.M.P. van der Aalst, A.P. Barros, A.H.M. ter Hofstede, B. Kiepuszewski, and B. Advanced Workflow Patterns, 2000.

[70] Arie van Deursen, Paul Klint, and Joost Visser. Domain-Specific Languages: An Annotated Bibliography. *SIGPLAN Not.*, 35(6):26–36, 2000.

[71] Eelco Visser. WebDSL: A Case Study in Domain-Specific Language Engineering. pages 291–373, 2008.

[72] J. E. White. RFC 707: High-Level Framework for Network-Based Resource Sharing, December 1975. Status: UNKNOWN. Not online.

[73] Wolfram Wiesemann, Ronald Hochreiter, and Daniel Kuhn. A Stochastic Programming Approach for QoS-Aware Service Composition. In *CCGRID '08: Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid*, pages 226–233, Washington, DC, USA, 2008. IEEE Computer Society.

[74] Petia Wohed, Wil M.P. van der Aalst, Wil M. P, Marlon Dumas, and Arthur H.M. ter Hofstede. Analysis of Web Services Composition Languages: The Case of BPEL4WS. In *Proc. of ER'03, LNCS 2813*, pages 200–215. Springer Verlag, 2003.

[75] Liangzhao Zeng, Boualem Benatallah, Anne H.H. Ngu, Marlon Dumas, Jayant Kalagnanam, and Henry Chang. QoS-Aware Middleware for Web Services Composition. *IEEE Transactions on Software Engineering*, 30(5):311–327, 2004.