Testing Service-oriented Architectures using a simulation

Bart van Teeseling

Rijksuniversiteit Groningen

Supervisors:	Alexander Lazovik Rijksuniversiteit Groningen
	Rix Groenboom Parasoft Netherlands bv
External Reviewer:	Paris Avgeriou Rijksuniversiteit Groningen

Contents

1. Abstract

The testing of a new component for an already running service-oriented system can often involve high costs, since it is hard to halt a service-oriented system with its often distributed components. Performing testing while the system is running can cause a performance decrease, pollution of the system with test data, and some service components might cost money per invocation.

In this paper we propose a method for using WS-BPEL to simulate a service-oriented architecture. Any new components that require integration testing can first be tested using such a simulation. Such a simulation uses locally deployed web services, and each web service can have simulated functionality for executing test scenarios. WS-BPEL is well suited for working with web services, and it allows some simple programming constructs, and therefore is a good candidate for being the programming language to create such a simulation in.

In this paper we create a simulation of a business process for a Dutch health law called WMO. By doing that, we show that it is feasible to create such a simulation, and we evaluate how well suited WS-BPEL is for the job.

2. Introduction

During a test session in 2009 at Pink Roccade test data was sent out by the service-oriented system while it shouldn't have been. As a result, twenty five people living in Kaag en Braassem in the Netherlands received notice that they were married, a certificate for their newborn, and some even received notice about their deceasing. None of this was true, but because the test data had propagated through the service-oriented system, the notices were sent out [26,29]. This small story is just an example of what can happen when you perform testing on an up and running service-oriented system. To prevent these kinds of things from happening, it is desirable to have a seperate testing environment. In this paper we evaluate a simulation implemented in WS-BPEL as such an environment.

Systems with a service-oriented architecture can be made up out of many different sub-systems and services that can be hosted in different places, and can be shared by many different users. When integrating a new entity or sub-system into an existing system with an service-oriented architecture, it can be difficult to test this new entity with respect to its use of other entities, services, and sub-systems (integration and acceptance testing).

When the system is used by multiple organizations it is not desirable to halt the system and to test the integration of the new sub-system. Halting the system would cost time and money, since the organizations can't make use of the existing system. If the system involves monitoring or operating critical real-life systems like a power plant, testing in the real-life environment could be dangerous too!

Some service providers charge money for service invocation. Also, since testing service-oriented applications often involves multiple invocations [2], the burdon on the system might cause a service provider not to meet its Service Level Agreements (SLA), and other users of the service might be bothered.

In chapter 3 we will further define the research questions of this thesis.

In chapter 4 we will look into related research, and explain how our work relates to that research.

In chapter 5 we explain important background concepts and techniques that play a big role in our work.

Chapter 6 presents a real life business process, and possible scenarios that can occur in that process. The business process of chapter 6 wil serve a case study for our poroposed solution in chapter 8. A WS-BPEL simulation for the use of integration testing without burdoning the real service-oriented system.

In chapter 7 we will present our proposed solution in a general form, without relating it to any specific examples.

Chapter 8 presents an implementation of the example business process presented in chapter 6. We use the outcome of this implementation for our evaluation of the porposed solution.

In chapter 9 we perform an evaluation of our solution and the role of WS-BPEL in it.

Chapter 10 shows points related to our solution that are interesting for further research.

Finally, chapter 11 presents a summary of our findings and some final conclusions.

3. Problem statement

To reduce the testing costs (time,effort, and money) for integrating an application into a service-oriented system, it would be desirable to find a solution for simulating the existing service-oriented system, so that the new component to be integrated can be tested using this simulation. This way, as much of the testing as possible can be performed locally in a simulation. By doing this we have to do less of the testing on the actual system.

3.1. *Simulated integration testing in service-oriented architectures*

In this thesis we propose the use of WS-BPEL for simulating a service-oriented system.

We think that WS-BPEL would be an appropriate language for writing and executing test scenarios. WS-BPEL can execute quite complicated processes and can be used to simulate the functionality behind the web-services of the existing service oriented system. Since WS-BPEL works well in combination with web services, we think it is the ideal candidate for this simulation task. However, we expect that WS-BPEL as a programming language also has limits. Therefore we want to assess how well WS-BPEL is suitable for this job, and what the important limitations of WS-BPEL are in this context.

3.2. *Research questions*

The general research questions we are interested in are ***"****How can we simulate integration and acceptance testing for new entities to be hooked up to an existing system with a service-oriented architecture using WS-BPEL?"* and *"What are possible limitations of WS-BPEL we encounter in this context?"*. We will evaluate the possibility of using BPEL by working out and simulating a scenario provided by local municipalities concerning a WMO law implementation, and by looking at the problems and limitations of BPEL we encounter. This leads us to the following specific research questions:

1. How can we create a simulation of a given service oriented architecture using WS-BPEL.
2. What are the problems and limitations of BPEL we encounter when we implement a system that answers questions 1.

We will try to answer question 1 by implementing a prototype for a given scenario and a given software-oriented architecture. We will answer question 2 by evaluating our implementation in the chapter called "Evaluation".

4. Related work

There is a lot of existing research on the testing of web services. There is not a lot of work though on decreasing the costs and problems of integration testing in an already functional service-oriented system. We discuss the major areas of web service testing, and how they relate to our work in this chapter.

4.1. *Test data generation*

There are researchers who focus on the generation of test data for web services. Some use the WSDL to test web services [4,7]. Others focus more on a specific class of input data for test generation, like for instance data that hackers might produce atacking a web service [5], or request messages that lead to execution failure [24]. Our research differs from this in that we focus more on composite web services, some of which are simulations of which we already assume they function correctly.

There is also research on test data generation for web service compositions[6]. This is actually quite related to our research. This method generates test data for a BPEL process of composite web services. We also use BPEL, but we use it to simulate existing web services to cooperate with a new web service that we wish to test. Actually the data generation method proposed in [6] might be used in combination with our research.

4.2. *Test automation frameworks*

There are a lot of researchers that propose to use some sort of method for test automation while testing web service or compositions of web services.

Often services' WSDL's are used to test services seperately. For instance [8] proposes to generate grammars from WSDL's as an intermediate step in generating a web service stub and a web service invocation generator. This way calls to a web service can be tested as well as the web service itself using generated web service calls. Other methods, like proposed in [10,11,12,13], analyze the WSDL to automatically generate web service calls, and check the web service's response.

In [9], the WSDL is analyzed in order to detect incompleteness or inconsistensies in a web service.

Other research focusses on special types of testing to uncover more complex type of errors, like for instance [14], where invariants are analyzed to see if they are violated. Types of errors uncovered here might be design errors instead of implementation errors. A service might need more checks, for preconditions for instance, than it has. Other research that focusses more on testing the web service model and not just things like the data format is [15], where ontology properties are tested with positive and negative test cases.

[16] Proposes a method to automatically determine the compatibility of web services. [17] proposes to enhance service compatibility by testing a service living up to its specifications before letting it register at a service discovery mechanism.
Our research also tests web services, but instead of using just the WSDL specification to analyze input and output, we analyze the web service's cooperation with its

simulated environment to see if it works well. Next to just performing correct calls and analyze responses, we also analyze the functional behaviour by having our simulation encorporate some semantically usefull behaviour.

4.3. *Contract based testing*

[18] and [19] propose to add contracts to a web service specification, to better express what can be expected of a web service, and what is needed to use that web service. By using the contracts one can test a service's validity. Since the contracts provide information about the inner behaviour of a web service, it could also be helpful in simulating a web service for a testing environment.

Instead of providing contracts, our method proposed to provide a simulation that a web service should cooperate with. If the simulation is good enough, a tester should be able to analyze the behaviour of other web services by watching the response of the simulated environment, and make sure that the tested service responses well to this behaviour. Of course contracts could be used in combination with our method, to describe the environment's behaviour more formally.

4.4. *Conformance Testing*

Some researchers propose methods of conformance testing web services. The idea is to increase the chance of succesful cooperation of web services, by ensuring that a web service behaves according to its specification, and by ensuring web services abide certain rules, like pre- and postconditions or invariants.
[17,21,28] propose ways to perform conformance testing, while [20] proposes to extend WSDL with information about a service's behaviour, which could be used in conformance testing.

Our work also performs a kind of conformance testing, in that we provide a simulated environment (a set of BPEL processes) of services, in which a service should succesfully operate. Instead of testing a service for following its specification, we test it for interoperability with the provided environment.

4.5. *Mock Objects*

Mock objects are a well-known technique to substitute parts of a program which are irrelevant for a particular unit test [22,23]. In [23] a technique is presented, that adds behaviour to mock objects. The behaviour is based on analyzing unit test cases. [3] also proposes the use of mock objects with behaviour, and here the mock objects technique is actually applied to service-oriented applications. The mock objects in [3] are semantic service stubs, that are used to test conformation to pre- and post conditions, and can generate exceptions.

Our work is very much related to the mock object research described above. Basically we use BPEL processes as mock objects. In [3] semantic service stubs, implemented in Java, are used instead of BPEL. We use BPEL to implement meaningful test cases, while [3] is more suited for automated testing of pre- and postconditions conformation.

4.6. *Service-oriented architectures and integration testing*

There is research that focusses on integration testing for web services. One could argue that mock objects, and to a lesser degree also contract based testing and conformance testing, also falls under integration testing. Since they are such specific methods, we decided to mention them sperately from general integration testing.

[28] presents a testbed for service-oriented architectures. Next to experimenting and testing ideas, such a testbed could be used very well for simulating an existing system and then integration testing of a part of that system. This is basically what we propose with our method using BPEL.

5. Background

A lot of concepts in this paper have to do with web services and techniques related to web services. In this chapter we explain some of the concepts and techniques that are involved in using web services and service-oriented applications.

5.1. *Web Services*

What exactly are web services? Perhaps the most clear and abstract definition can be found on [31]:

"*Web services are typically application programming interfcaes (API) or Web API's that are accessable through HTTP and executed in a remote system hosting the requested services*"

Another formal definition given by [30] is:

"*A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.* "

Another way of looking at web services is as a collection of standards, of which the most important ones are SOAP, XML, WSDL, and UDDI [28].

So how do web services work? Different platforms and programming languages can exchange messages (like function calls and replies) through a language called XML. The prescribed format of these messages is called the "*Simple Object Access Protocol*" (SOAP).
A web service has a description of its interface, so that clients can see how to consume a web service. This interface is written in an XML-based language called "*Web Service Description Language*" (WSDL).
There is a way for centrally registering you web service so others can find it, which is called "*Universal Description, Discovery and Integration*" (UDDI).

In this paper it is important to know the concepts Web Service, XML, SOAP, and WSDL.

5.2. *Service-oriented Architecture*

What is service-oriented architecture? A formal definition can be found in [33]:

"In computing, a service-oriented architecture (SOA) is a flexible set of design principles used during the phases of systems development and integration. A deployed SOA-based architecture will provide a loosely-integrated suite of services that can be used within multiple business domains."

In [30] a Service Oriented Architecture is defined as a form of a distributed systems architecture that:

- is an abstracted, logical view of the actual program, and is defined in terms of what it does, and typically carries out a business-level operation.
- is message orientated; There is no need for knowledge about the internal workings of a seperate service.
- uses services that have a formal description that can be processed by a machine, and the service semantics should be documented by this description.
- tends to use services over a network.
- is platform neutral. Messages have a platform-neutral format, usually XML.

Less formally defined, we can say that a service-oriented architecture is a form of distributed computing, where an application is made up of several services. The business logic of an application is divided into different services that make up the total architecture of the application. The services that make up the application are usually distributed over different servers and work together over the internet or an intranet.

An important characteristic is that the services used in a service-oriented architecture are loosely coupled. The interfaces of the services are independent of the implementation [32]. Programmers can build an application by combining several services, without having to know the inner workings or exact implementation of these services.

Most often service-oriented architectures use the web service standards described in the previous section on “web services”. In the remainder of this paper, when we refer to a service-oriented architecture, we assume it used the web service standards, like SOAP and WSDL.

5.3. *Business Process*

In this paper we mention the term “business process”. Although it is hard to assign an exact universal definition to the term, the concept is not very complicated. Formal definitions could be ‘The set of internal activities performed to serve a customer’ [34], or ‘Set of partially ordered activities intended to reach a goal’ [35].

We would like to define a business process as “a set of related activities that together serve a goal (like a service or a product) for a customer or customers”. A business process can often be visualized with a flowchart. We derived this definition from [36].

5.4. *BPEL*

BPEL stands stands for “business process execution language”, and it is short for WS-BPEL, which stands for “web service business process execution language”.

WS-BPEL is used to describe how a business process behaves. This includes interactions between a process and its partners. An interaction with a partner only occurs by means of Web Service interfaces. WS-BPEL orchestrates the service interactions of a process with its partners to achieve some sort of business goal [37].

Less formally put, BPEL is a language used to orchestrate web services. It defines how the web services used will work together to complete the desired business process. BPEL is a programming language based on XML. Often programming in BPEL is done using a graphical editor, and not just by typing in BPEL by hand [38]. In this paper also a graphical interface tool is used to progrma in WS-BPEL. The tool used is BPEL Maestro.

For this paper it sufficient to understand that BPEL defines processes that orchestrate web services, and that information import and export in BPEL is achieved exclusively through web service interfaces.

To give a better idea, below you will find a screenshot of a graphical representation of a BPEL process implementing a marketplace process.

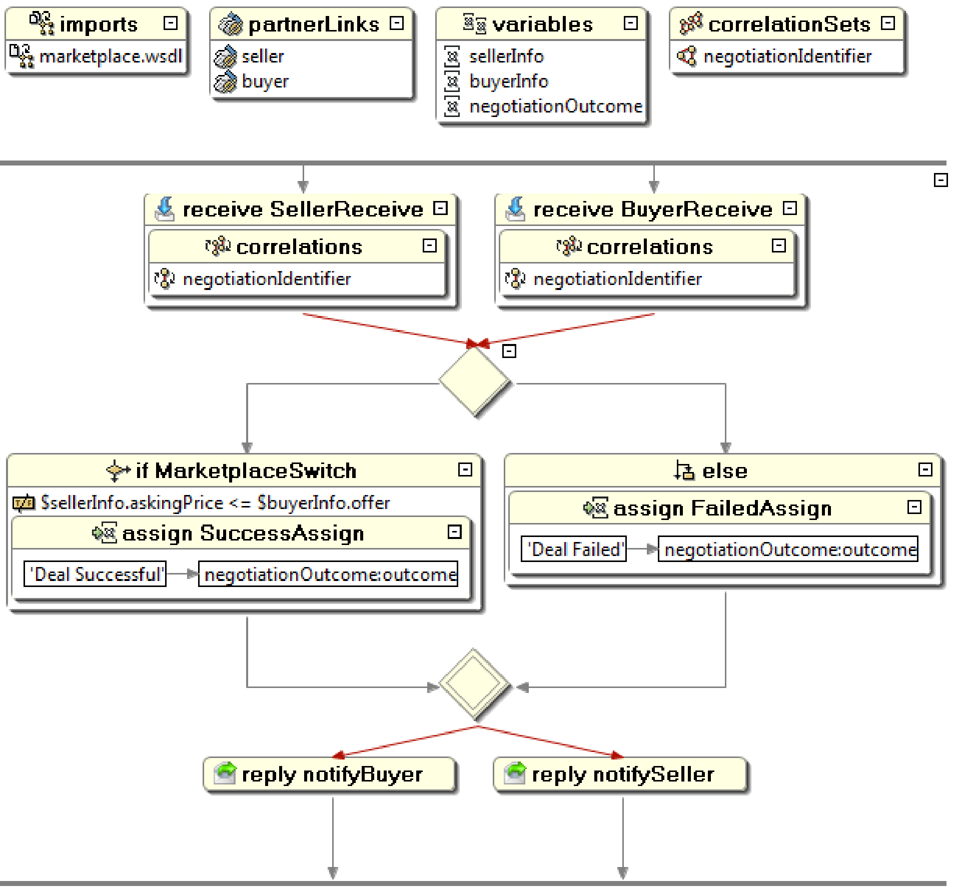
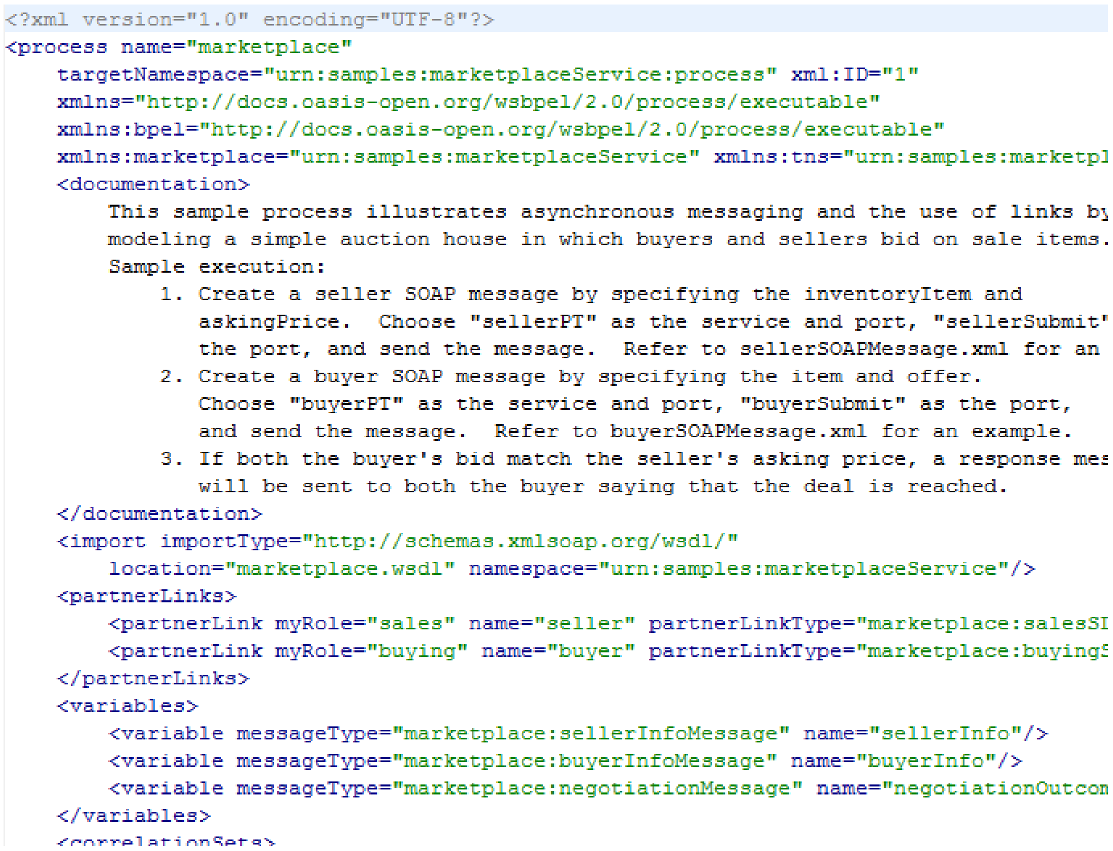


Figure 1: Graphical representation of a BPEL process in BPEL Maestro software.

Part of the code for this process looks like:



```
<?xml version="1.0" encoding="UTF-8"?>
<process name="marketplace"
    targetNamespace="urn:samples:marketplaceService:process" xml:ID="1"
    xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
    xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
    xmlns:marketplace="urn:samples:marketplaceService" xmlns:tns="urn:samples:marketpl
    <documentation>
        This sample process illustrates asynchronous messaging and the use of links by
        modeling a simple auction house in which buyers and sellers bid on sale items.
        Sample execution:
            1. Create a seller SOAP message by specifying the inventoryItem and
               askingPrice.  Choose "sellerPT" as the service and port, "sellerSubmit"
               the port, and send the message.  Refer to sellerSOAPMessage.xml for an
            2. Create a buyer SOAP message by specifying the item and offer.
               Choose "buyerPT" as the service and port, "buyerSubmit" as the port,
               and send the message.  Refer to buyerSOAPMessage.xml for an example.
            3. If both the buyer's bid match the seller's asking price, a response mes
               will be sent to both the buyer saying that the deal is reached.
    </documentation>
    <import importType="http://schemas.xmlsoap.org/wsdl/"
        location="marketplace.wsdl" namespace="urn:samples:marketplaceService"/>
    <partnerLinks>
        <partnerLink myRole="sales" name="seller" partnerLinkType="marketplace:salesSI
        <partnerLink myRole="buying" name="buyer" partnerLinkType="marketplace:buyingS
    </partnerLinks>
    <variables>
        <variable messageType="marketplace:sellerInfoMessage" name="sellerInfo"/>
        <variable messageType="marketplace:buyerInfoMessage" name="buyerInfo"/>
        <variable messageType="marketplace:negotiationMessage" name="negotiationOutcom
    </variables>
    <correlationSets>
```

Figure 2: Example of BPEL code.

A typical BPEL file will contain the following:

- **A section of partnerlinks**: Here the relationships are defined. What web services does the process work with, and what is their role and our role in the business process.
- **A section with variables**: The data structures that will be used throughout the process activities.
- **Faulthandlers**: Handlers that become active when any errors occur.
- **A sequence**: The program logic for the business process, including service calls, replies, loops, parallel flows et cetera.

5.5. *Integration testing*

In this paper we focus on the problem of integration testing of components that are newly added to an already existing service-oriented system. To be clear and complete we give our definition of integration testing here.

Integration testing is the problem of putting a system together out of seperate components and testing the resulting systems for any problems that are a result of the interaction between the seperate components [39]. In this paper we have the problem of a functioning service-oriented system and we want to add a new component to it. After that we want to test if the new component works well together with the already existsing service-oriented system.

6. Example: WMO service-oriented system

To illustrate our problem and why we propose to evaluate our particular solution we have chosen an example of a service-oriented to simulate with BPEL. In this section, first we will explain what the problem is with integration testing of a service-oriented system, and why this cannot be solved well in a traditional way of testing.

After this we will explain some background on the business process behind the example service-oriented system. Also, we will describe the service-oriented architecture that is to be simulated itself. Based on this architecture a set of scenarios will be defined.

After that we will describe the design and implementation of the actual prototype that implements the simulation of the service oriented architecture in the next chapter.

6.1. *The problem of integration testing a service oriented architecture*

In a non-service-oriented system we can solve the problem of integration testing in a traditional way. In a service-oriented system we cannot. This is because a service-oriented system is already running.

In a traditional system, we control the whole system. We integrate a new component, we test the system with test data, and we can look throughout the whole system what happens. We can make the test data not go into the database, to prevent our real database from getting polluted with test data.

In a service-oriented system, we have problems that we don't have in other type of systems:

- Our system components are distributed among different servers at different locations.
- Our system components might be maintained by different owners or companies.
- Our system components might be written in completely different programming languages than we use ourselves, running on completely different platforms.
- Our system components are acting as an active component in an up and running system, while we perform the integration testing.

So we can't just stop our system to perform the integration testing of a new component, since we don't control all the components. We also cannot put our system in some sort of debug state to prevent our database and processes from getting polluted with test data.
Imagine that we are working on integrating a component for a hospital system. We can't just insert test data saying that someone is dead or sick while the system is running in the mean time. This would cause major inconsistensies and problems with a persons insurance. This is just one small example of what could happen when we test a component after integrating it in a service-oriented environment.

The problems described above can't be solved completely. We will have to somehow test a new component after integrating it in a service-oriented environment. We can however try to make sure that our component has the least amount of errors possible, before we integrate it, and try to perform as much of the testing “offline” (not in the real system), before we integrate the component and do the final testing. If we can keep the integration testing to a minimum at least we minimize the problems. To this end, we want to simulate the service-oriented environment for the component, and perform as much of the integration testing as possible in this simulated environment. In this paper we will work out such a simulation in WS-BPEL to assess how well WS-BPEL would be suited for this job. In the next section we introduce an example of a service-oriented system. This system will be simulated using WS-BPEL in the chapter after that. Using the implementation of that simulation we will try to evaluate how well WS-BPEL is suited for simulating service-oriented systems for the goal of “offline” integration testing.

6.2. *WMO business process*

To evaluate the use of WS-BPEL as a means for simulating a service-oriented system we will implement a simulation of the WMO business process.
WMO (Wet Maatschappelijke Ondersteuning) is a Dutch law for supporting people that have a chronic disease or disability, so that these people can independently live in their own homes and actively take part in everyday life despite their physical problem. Support provided by the WMO are typically things like transportation (e.g. taxi transportation), a wheelchair, or a home modification (e.g. removing door posts for people in a wheelchair). The responsibility for a WMO request lies with the Dutch municipalities, they handle the complete business process. The process does access external parties like for instance insurance companies and doctors for medical advice.

In the figure below you see a description of the service-oriented architecture for requesting WMO. It is a simplification of a real life situation based on the situation in the town of Haren in the Netherlands [1].

The various activities in the business process are represented by web services, as described in our definition of service-oriented architectures in the section “Background”. First, a municipality is contacted by a citizen requesting some kind of support, like a wheelchair for instance. If the request is correct a medical advice is requested from a doctor by means of a service invocation. If the doctor's office (GGD in this Dutch business process) replies with a positive advice, the business process is carried on. Otherwise, the request is declined. After a positive advice, depending on the type of care requested, the insurance company is contacted through a service to determine the amount of the cost of the requested care that is covered by the citizen's insurance. The insured amount is deducted from the price, if the costs are not completely covered the rest of the cost are billed to the citizen.
The implementation or care supply is requested though a service, which replies whether or not the care will be delivered succesfully.

This type of architecture allows for new components to be hooked up. For example a new care supplier, a new doctor's office (GGD), a new insurance company, or a complete new municipality office could be hooked up. This new component would have to be integrated with the current active service-oriented system, and that is where our integration testing problem arises. In the next chapter we will implement a simulation of the service-oriented WMO system that was described above, and we will explain how that could be used for “offline” integration testing. The chapters after that will evaluete how well suited this method is.

6.3. *WMO architecture and scenarios*

In the previous section we described the WMO business process, in this section we will depict the WMO business process and describe the possible scenarios that can arise in that business process.

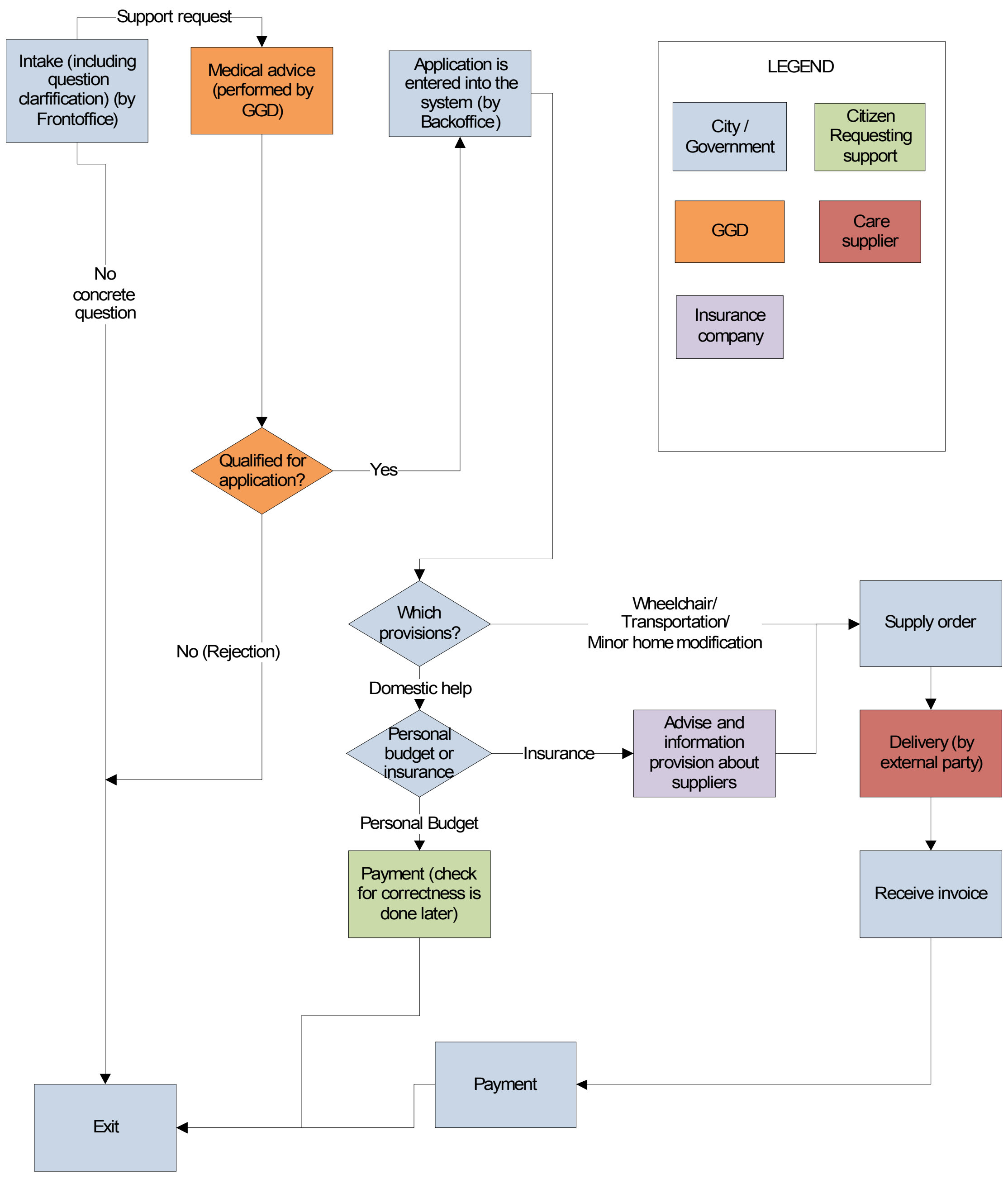

Support request
Intake (including question clarfification) (by Frontoffice)
Medical advice (performed by GGD)
Application is entered into the system (by Backoffice)
LEGEND
City / Government
Citizen Requesting support
GGD
Care supplier
Insurance company
No concrete question
Qualified for application?
Yes
No (Rejection)
Which provisions?
Wheelchair/ Transportation/ Minor home modification
Supply order
Domestic help
Personal budget or insurance
Insurance
Advise and information provision about suppliers
Delivery (by external party)
Personal Budget
Payment (check for correctness is done later)
Receive invoice
Payment
Exit

Figure 3: Business process for the Dutch WMO law

In the architecture communication between different parties (indicated by different colors) is done through web services. For each of these parties we can define integration testing scenarios.

Government integration testing scenarios

- Receiving WMO request from citizen, care is supplied, care is completely covered by insurance
- Receiving WMO request from citizen, care is supplied, care is not or partly covered by insurance
- Receiving WMO request from citizen, negative doctors advice, care not supplied.
- Receiving WMO request from citizen, bsn unknown, care not supplied.
- Receiving WMO request from citizen, care is unknown by supplyer care not supplied.

GGD integration testing scenarios

- Sending medical advice to Government
- Receiving medical advice request from Government

Care supplier integration testing scenarios

- Receive supply order from Government
- Send result to government, supply care.

Citizien integration testing scenarios

- WMO request to Government (Through web form for example)
- Receive result and instructions from government

In a service-oriented architecture like this, made up of different independent parties, it might happen that a new party needs to be connected to the system.
For instance, imagine a new town office to be opened for a new city, or a city that wasn't hooked up to the WMO service system yet. In order for the town office to be hooked up to the WMO system it would have to test its software. To do this, it would have to call the services for “medical advice” and “supply order” repeatedly. When these service calls cost money per seperate call this can cost a lot of money. Also, the overall performance of the WMO system can degrade. Next to that, employees at the offices of the order suppliers and the medical advice office need to weed out the test cases from the real cases. This costs the employees time, and therefore this costs their employer money. So by testing a new part of a service-oriented system we stand to loose a lot of performance, time and money.

7. Integration testing using a WS-BPEL simulation

In general we can define the problem we have when we want to perform integration testing after integrating a new component into an already exisiting and actively running service-oriented system as follows:

Current situation:

1) We have a service-oriented system, which means we have a system that is composed of different components represented by web services. The different components of the service-oriented system are distributed over different locations, and are in control of different stakeholders, parties, or even different companies. The system is up and running and actively used bu users.
2) We have a new component that must be connected to the running service-oriented system. Assuming the component has been tested seperately (e.g. by unit testing), integration testing has to be performed.

Problems:

1) The system is running and it's actively being used. When test data is used while performing the integration testing, the system can get polluted with test data. Test data could be saved in files or data bases where it would get mixed with real data. The users of the system can't see the difference between legitimate system activity, and integration test activity.
2) If during the integration testing a lot of errors are uncovered, or if it is not clear what was the cause of a particular error, a lot of runs of the integration tests might be required. This can cause a decrease in overall system performance, which will be a problem for all system users. Also, some services cost money per invocation, so the more calls that are needed during the performance of the integration testing, the higher the cost of the integration testing can turn out to be.

In order to decrease the amount of live testing a new service when integrating it into an existing service-oriented system, we propose to create a simulation of the existing service-oriented system that can serve as an initial testing environment. The idea is that when you test a service in this simulated environment first, you minimize the amount of testing errors during live testing, hereby saving the costs of service invocation, employees' wasted time, and minimizing the overall system performance degradation. In other words the goal is to decrease the overall cost of the process of integration testing.

So what do we mean by a WS-BPEL simulation of a service-oriented architecture? To be precise, we mean to simulate both the business process as well as the services it is composed of. We want to implement such a simulation using WS-BPEL files. One WS-BPEL file to orchestrate all the services to implement the business process, and one WS-BPEL file for each web service component. We want to give each web service component (which is a WS-BPEL file in the simulation) hardcoded functionality, so that we can perform some usefull actions during integration testing, using our simulation.

To illustrate the solution lets first define the general situation.

- We have a live system with a service-oriented architecture.
- The system is made up of several services, and multiple parties can provide the same service (for instance, multiple doctors can be hooked up to be called for medical advice, or a town could have multiple office's where one could go to for WMO requests, et cetera).
- A new party wants their service to be connected to the service-oriented system. This service needs to be tested before it can be integrated succesfully.

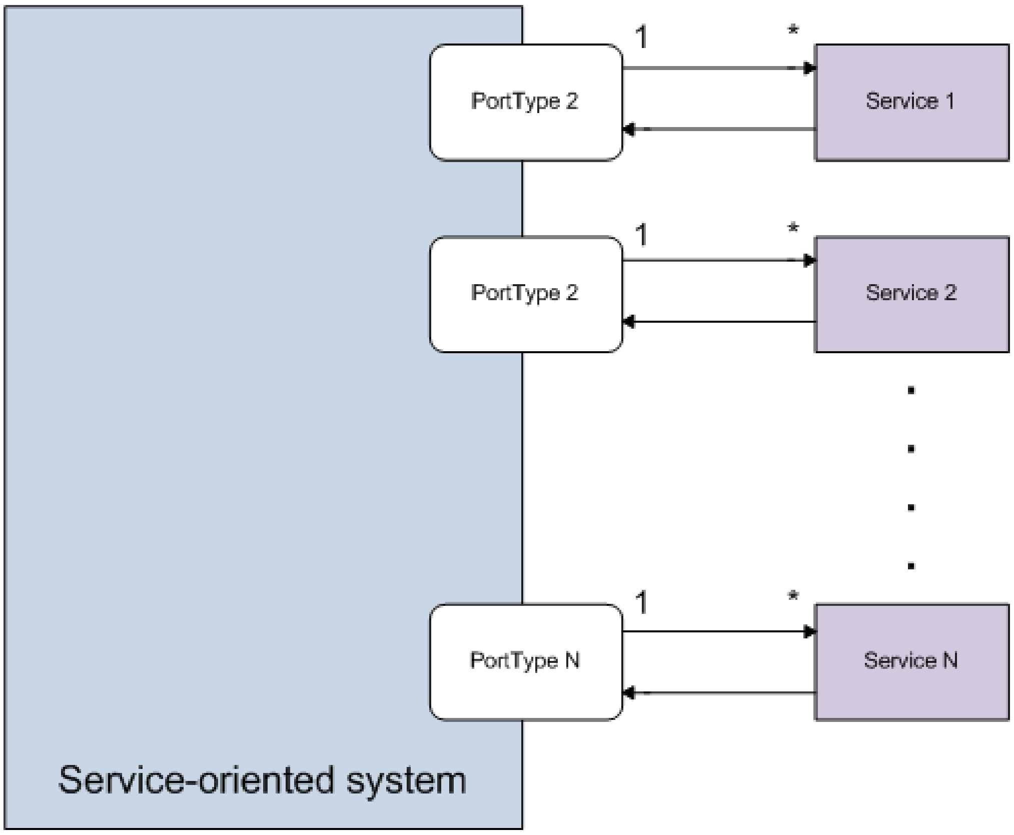


Figure 4: General structure of a service-oriented system.

7.1. *Using a simulation of the service-oriented architecture*

Now when a new party wants to connect its service to the system it has to be tested. To do this calls have to be made to the new service from the system, and the other way around. The more errors the new service application has, the more the tests have to be repeated, and the live system gets burdonned with test data, mixed with real data from the already running services. To minimize this, we can simulate the service-oriented system, test the new service application in this simulated system, and while doing this catching as much errors as we can, before we move on to the actual live testing with the real service-oriented system.

So now we need a simulation of the whole system:

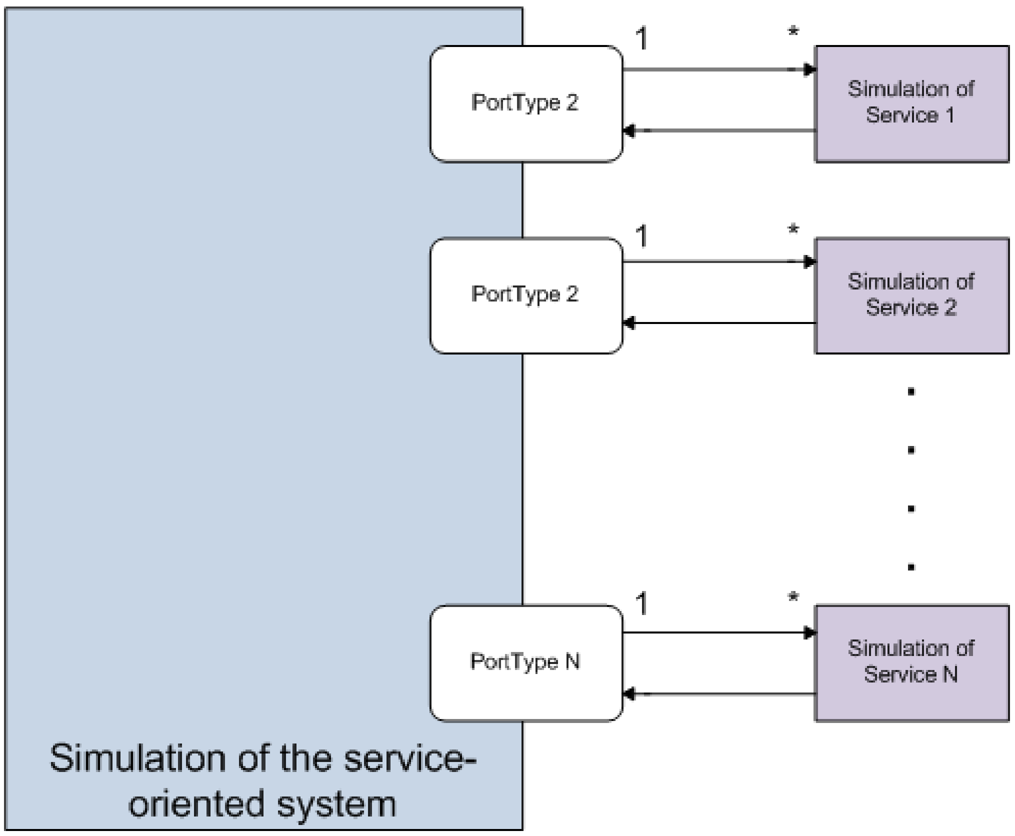


Figure 5: General structure of a simulation of service-oriented system.

The system administrator of the live service-oriented system can now distribute a version of the simulated system among everyone who wants to hook up a new service to the live service-oriented system.

For instance, when a party, lets call them company A, wants to connect an implementation of service number 2, they first test their application by taking the simulation of the system, taking out the simulation of service number 2, and replacing it with their own application.

7.2. *BPEL for functionality and creating test data and scenarios*

We propose to use BPEL for implementing the system simulation. By using BPEL we can add simple program logic to orchestrate the web service. Also we can add some simple behaviour to each of the simulated services. To illustrate this consider an example:

For simulating service N, part of the whole simulated service-oriented system, we use a seperate BPEL file. Lets say for instance, service N is a service that returns someone's name and address when you call it with that person's social security number. The service returns either the name and address when the person is a known citizen in the database, or some sort of exception when the social security number is unknown.
Now we can hardcode the simulation of service N to return a certain name and address when we call it with social security number “100001”, for instance “John Doe, Willowroad 1, Zip code 123, Manchester”. We can additionally hardcode the

service simulation to return either an exception or a "person-not-found"-message when we call it with social security number "100002".
We can do something like this for each of the service simulations. This way we can provide any testers with test data for all kinds of scenarios. They can call the simulation of service N with "100001" as an argument to test a scenario involving a succesful "name and address"-retrieval, or they can call it with "100002" to test a scenario involving an unsuccesful "name and address"-retrieval.

In addition to the adding of test data in the way described above, BPEL also provides easy means for checking the format of arguments involved in service invocations. So we can easily check whether an argument is of the correct format (string, integer, et cetera), and return an exception or error message otherwise.

Also now, the system administrator could distribute the simulation with a BPEL engine that can run the BPEL files that implement the simulation. Using such an engine a tester could just start the engine and he has an actual simulation running locally, without having to configure or program something. This brings us to the following situation for the simulation:

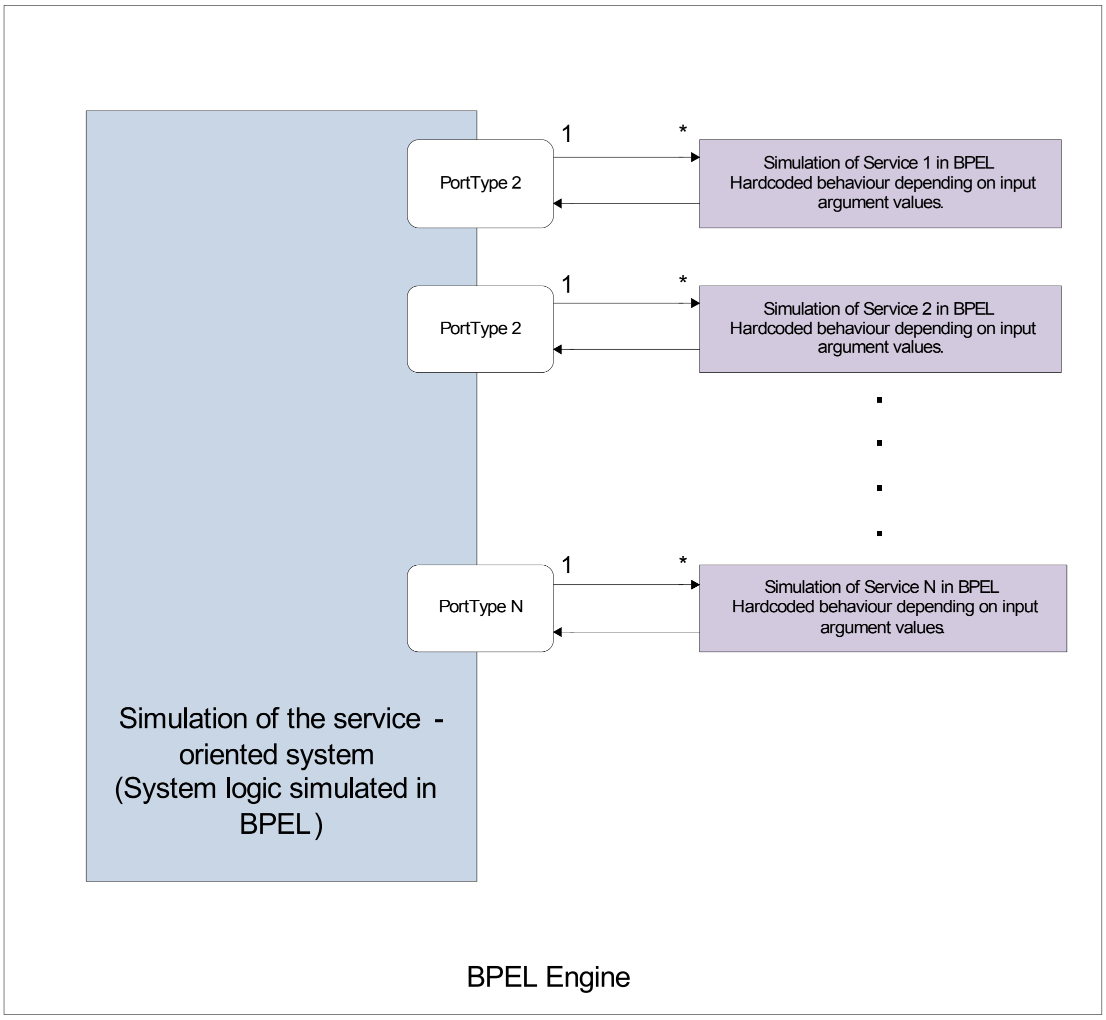

1
*
PortType 2
Simulation of Service 1 in BPEL Hardcoded behaviour depending on input argument values.
1
*
PortType 2
Simulation of Service 2 in BPEL Hardcoded behaviour depending on input argument values.
1
*
PortType N
Simulation of Service N in BPEL Hardcoded behaviour depending on input argument values.
Simulation of the service - oriented system (System logic simulated in BPEL)
BPEL Engine

Figure 6: General structure of a BPEL simulation of a service-oriented system.

So lets assume some party wants to create an implementation of service number 2, and wants to hook it up to the service-oriented system. What are the steps they have to go trough?

1. Contact the system administrator, who sends over a simulation of the system, together with test data.

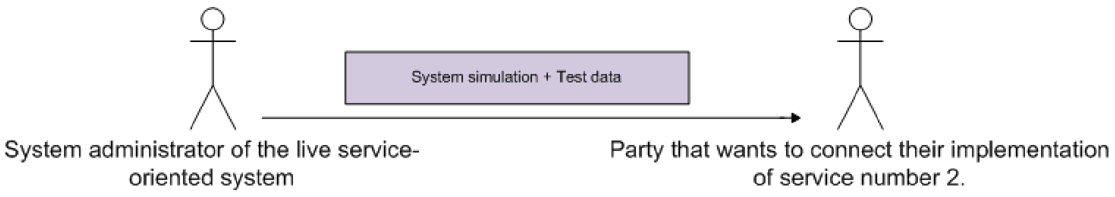

System simulation + Test data
System administrator of the live service-oriented system
Party that wants to connect their implementation of service number 2.

Figure 7: Send a simulation to those who want to connect their component to the SOA.

2. Take the simulation, take out the simulation of the part they want to test for, and replace it with their own service application.

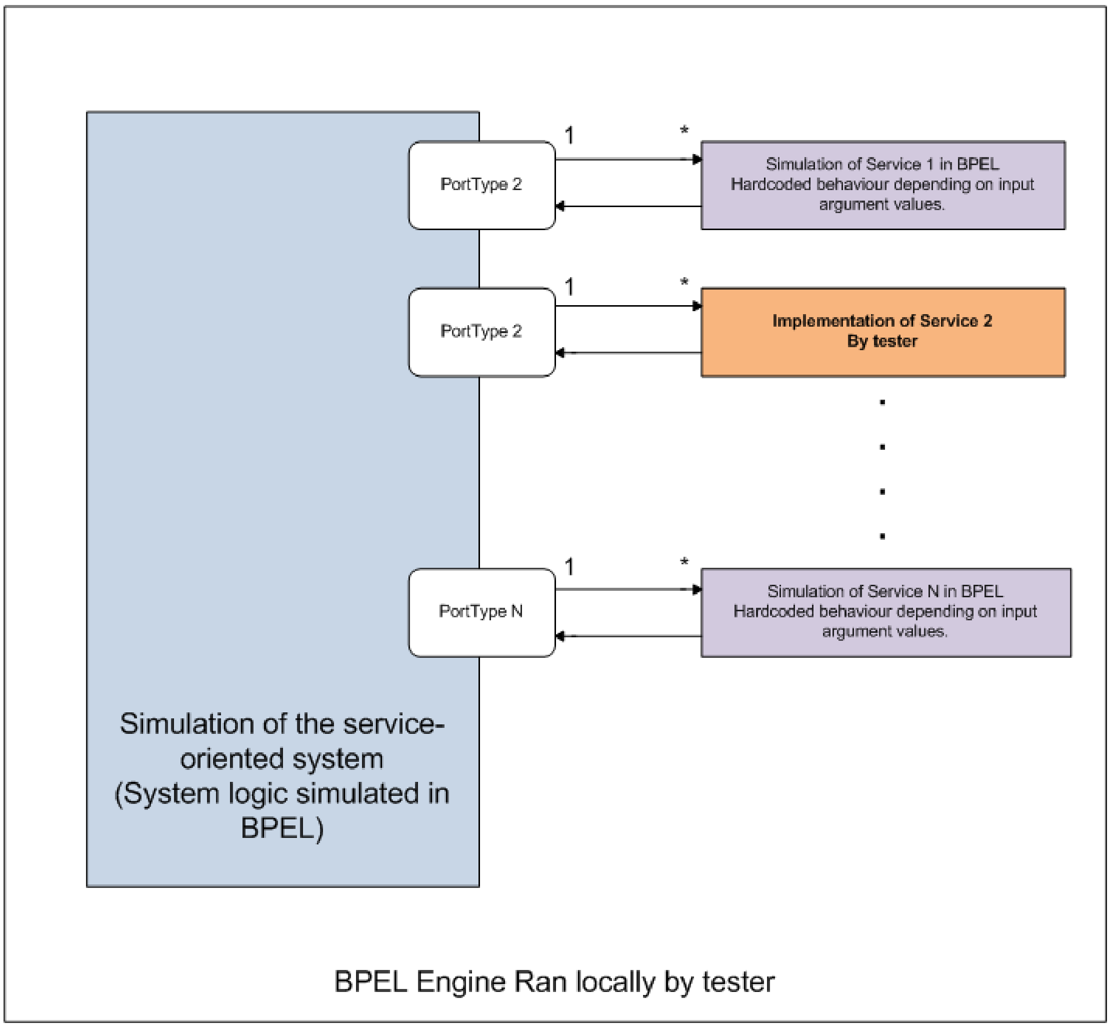

PortType 2
1
*
Simulation of Service 1 in BPEL Hardcoded behaviour depending on input argument values.
PortType 2
1
*
Implementation of Service 2 By tester
PortType N
1
*
Simulation of Service N in BPEL Hardcoded behaviour depending on input argument values.
Simulation of the service-oriented system (System logic simulated in BPEL)
BPEL Engine Ran locally by tester

Figure 8: In the simulation replace part of the simulation by your implementation.

3. Test their application integrated with the simulation, until it works well for all the test data.
4. Show the results of the simulation tests to the system administrator and get access for live testing.

8. Case Study: Implementation of WMO example

To give a demonstration of our approach, we give an implementation of the business process for WMO law, as discussed in chapter 5. We will use this implementation for demonstrating our proposed solution, and to evaluate that solution.
First we will describe the tools we used for implementing the case study, and then we will describe the design of the implementation, along with a description of the final resulting software.

8.1. *Tools and software*

We use the Parasoft tool SOAtest and BPEL Maestro. BPEL Maestro is used as the BPEL engine, and SOAtest is a testing tool among which web service calls can be generated. The reasons these tools were chosen are the following:

- SOAtest and BPEL Maestro work well together.
- BPEL Maestro is a BPEL Engine with an easy-to-use graphical user interface.
- Both tools were provided for free by Parasoft Netherlands b.v.
- SOAtest can generate web service calls which is very useful during testing of the BPEL scripts.

Below we will give a more elaborate description of the used tools, SOAtest and BPEL Maestro.

8.2. *SOAtest*

SOAtest is developed by a company called Parasoft, and it is a testing tool for service-oriented architectures. It contains features such as web service stubbing, mechanisms for regression testing of web services, and many other features that are beyond the scope of this implementation.
In our approach we only use SOAtest to test our separate BPEL files, by calling it and examining the response. So we use it to test our service-oriented system simulation or parts of it.

8.3. *BPEL Maestro*

BPEL Maestro is integrated with SOAtest, and also developed by Parasoft. BPEL Maestro is the tool we have used the most for this prototype implementation. We use it as a BPEL engine.
With BPEL Maestro we have a graphical editor for creating and editing WS-BPEL files. The tool allows us to switch back and forth between WS-BPEL code files and a graphical represntation of the WS-BPEL files. Besided editing and debugging, BPEL Maestro also provide BPEL engine, which allows us to actually run our BPEL files, so we can actively use them.

8.4. *Design*

We want to implement a BPEL simulation of the WMO business process, as depicted in figure 1.
We want to create seperate WS-BPEL files for each entity that would be a web service in real life. This way each component (web service) can be tested with the prototype. If you want to perform integration testing for a particular component, just

exchange the WS-BPEL file for the real implementation in the simulation. The seperate web service components will be orchestrated using one BPEL file that will contain all the components. This way a BPEL-simulation as a whole can be distributed, and testers can choose which part of the simulation they want to replace by their own application. In figure 9 you can see the high level design for the simulation of the implementation of the WMO business process WS-BPEL simulation.

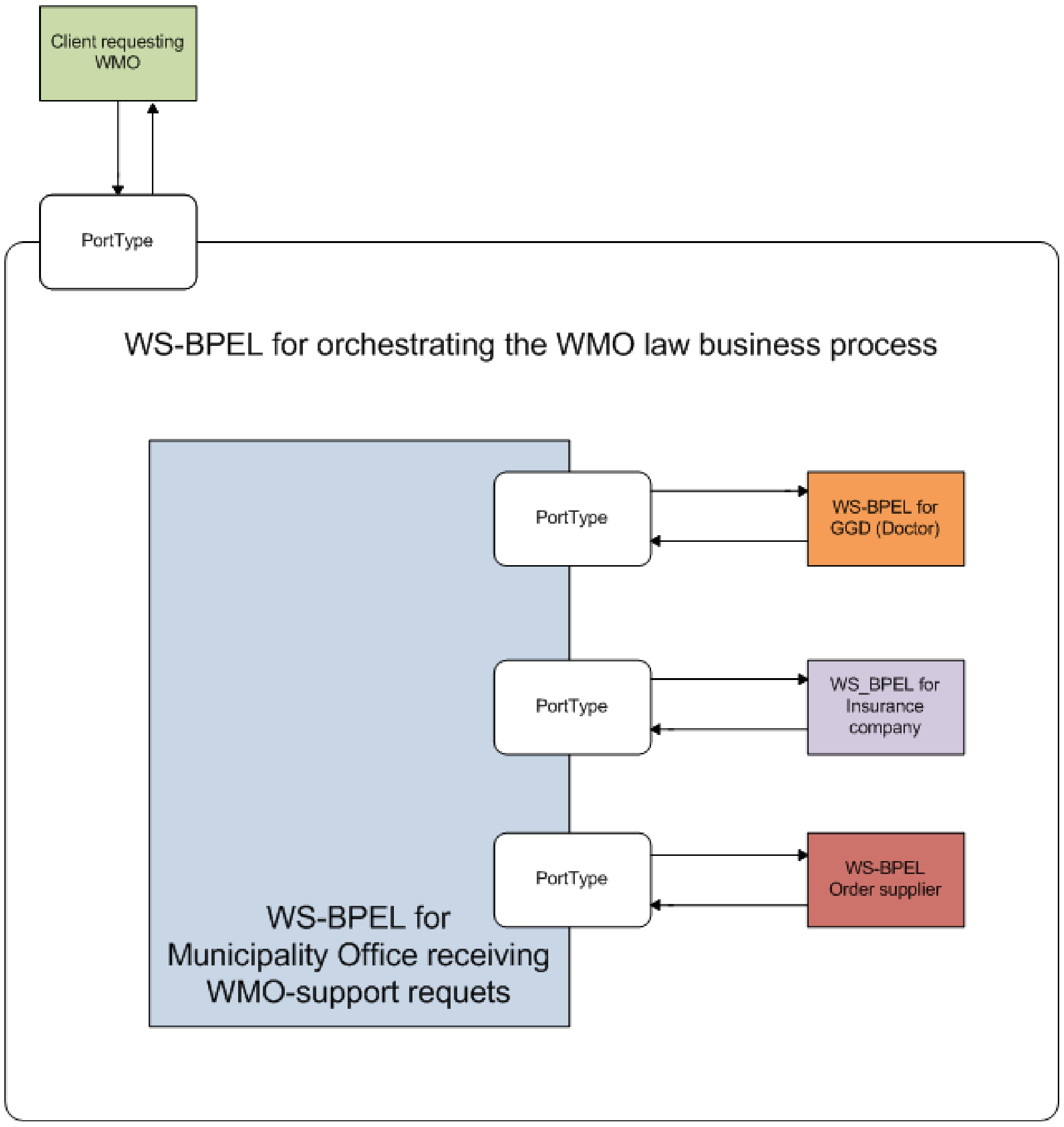

Client requesting WMO
PortType
WS-BPEL for orchestrating the WMO law business process
PortType
WS-BPEL for GGD (Doctor)
PortType
WS_BPEL for Insurance company
PortType
WS-BPEL Order supplier
WS-BPEL for Municipality Office receiving WMO-support requets

Figure 9: High level design for the WS-BPEL simulation of the Dutch WMO business process.

To illustrate the use of the solution lets say we want to implement an application for a municipality office taking in requests of citizens for WMO-support. We would go throught the following steps.

1) Receive the WS-BPEL-simulation of the whole system.

2) Replace part of the simulation that represents the municipality office by our own application of the municipality office service.

3) Locally test our application using the simulation, improve our application based on these results, until we find no more errors in our application using the simulation.

4) Integrate our application of the municipality office service with the real system.

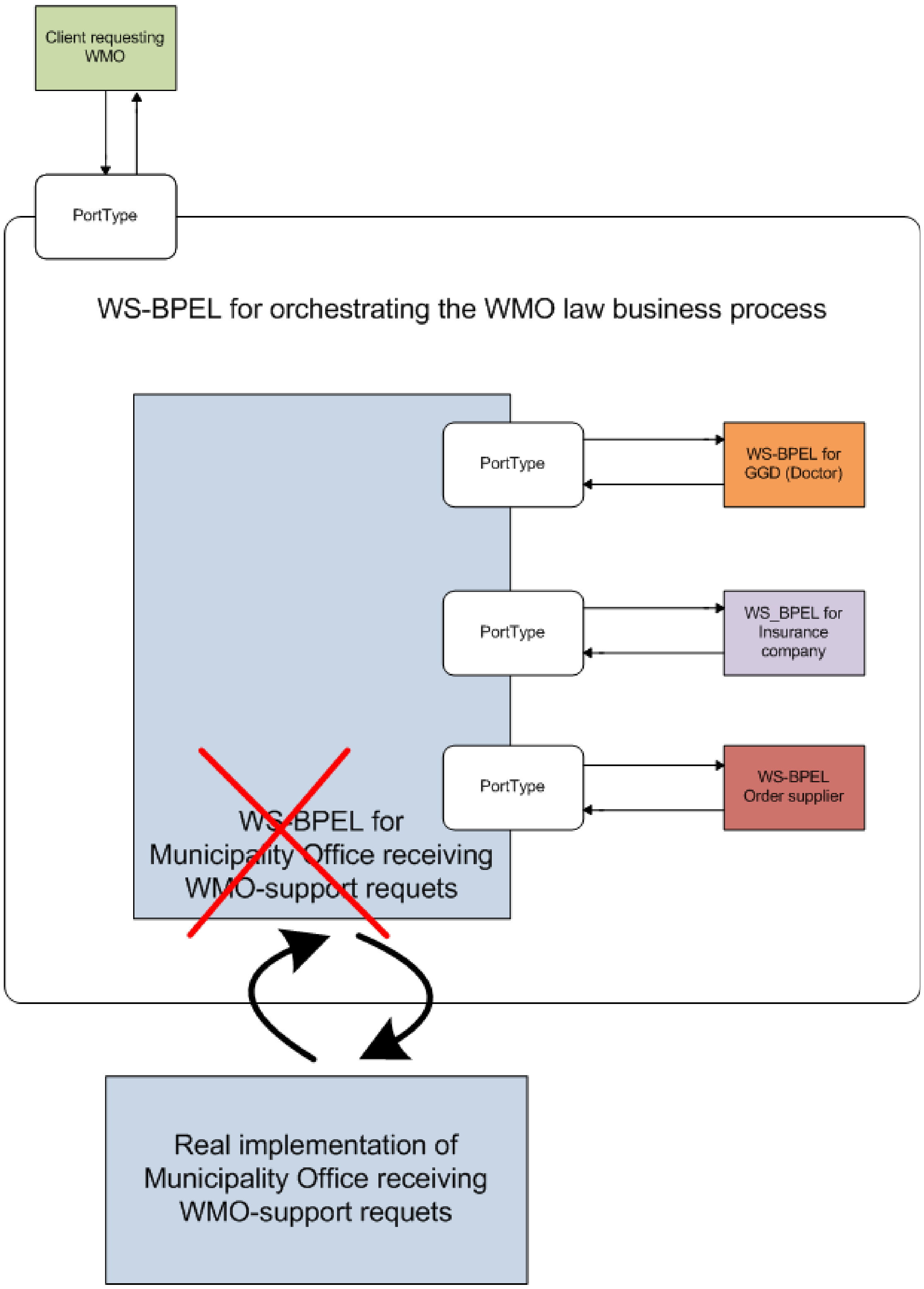

Client requesting WMO
PortType
WS-BPEL for orchestrating the WMO law business process
PortType
WS-BPEL for GGD (Doctor)
PortType
WS_BPEL for Insurance company
PortType
WS-BPEL Order supplier
WS-BPEL for Municipality Office receiving WMO-support requets
Real implementation of Municipality Office receiving WMO-support requets

Figure 10: How to use the proposed solution for integration testing of a municipality office implementation.

8.5. *Presentation of software*

For each of the seperate services, the Office receiving WMO-support requests, the GGD, the Insurance company, and the Order supplier we created a BPEL file. These

WS-BPEL files are contained in an overall WS-BPEL file which serves as the orchestrating file of the overall business process, like explained in the previous section on design.
For each of the WS-BPEL files, we will give a graphical representation of the file, which is a screenshot of the actual implementation, taken from the BPEL Maestro BPEL engine. Also we will give a graphical representation of the WSDL interface, which describes the web service interface of the WS-BPEL file. For each BPEL file we give an inpunt and output table, to describe input and output data for test scenarios. The input data can be used to execute test scenarios, and the output data describes what response to expect.
We will start with the seperate services that play a role in the business process, and end with the description of the BPEL file that represents the overall business process.

8.6. *GGD BPEL simulation*

The GGD is a doctor's office that can give medical advice about whether or not a client needs a certain kind of WMO support.

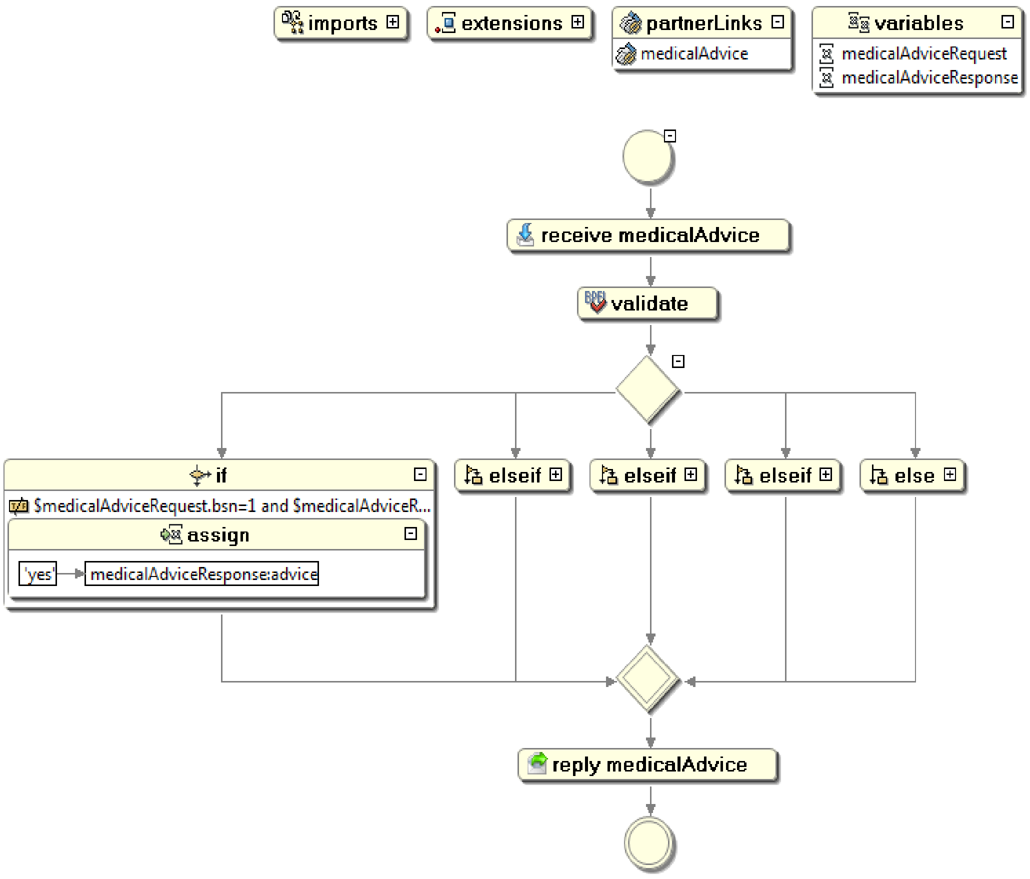
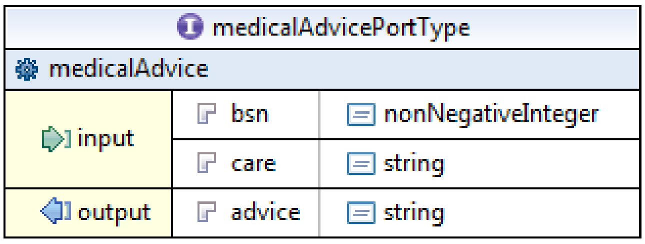


Figure 11: GGD Medical advice simulation in BPEL.

The graphical description of the service's WSDL:



medicalAdvicePortType		
medicalAdvice		
input	bsn	nonNegativeInteger
	care	string
output	advice	string

Figure 12: The graphical description of the service's WSDL.

BSN: The social security number of a person.
CARE: The type of care the person requested.

ADVICE: Whether the doctor thinks the WMO should be given {yes|no|unknown}.

We have implemented the doctor as a “medical advice” service in BPEL.
First of all we validate the input using the BPEL validate tag. This way we make sure the service is called with the correct data types.
By adding some simple functionality we were able to simulate some scenarios. By using some simple if-statements, we were able to make the medicalAdvice-service behave in different ways depending on the input. We show the service behaviour using a simple table:

BSN	**Care**	**Advice**
1	transportation	yes
1	aspirine	yes
1	Other than 'transportation' or 'aspirine'	no
2	wheelchair	yes
2	Other than 'wheelchair'	yes
Other than '1' or '2'	Any string	unknown

In this table you can see we can execute (part of) different scenarios by sending the service different input values.

8.7. *Insurance company BPEL silmulation*

The BPEL service for the insurance company is called “getInsurance”. It takes the BSN of a person and returns the maximum amount that person is insured for.

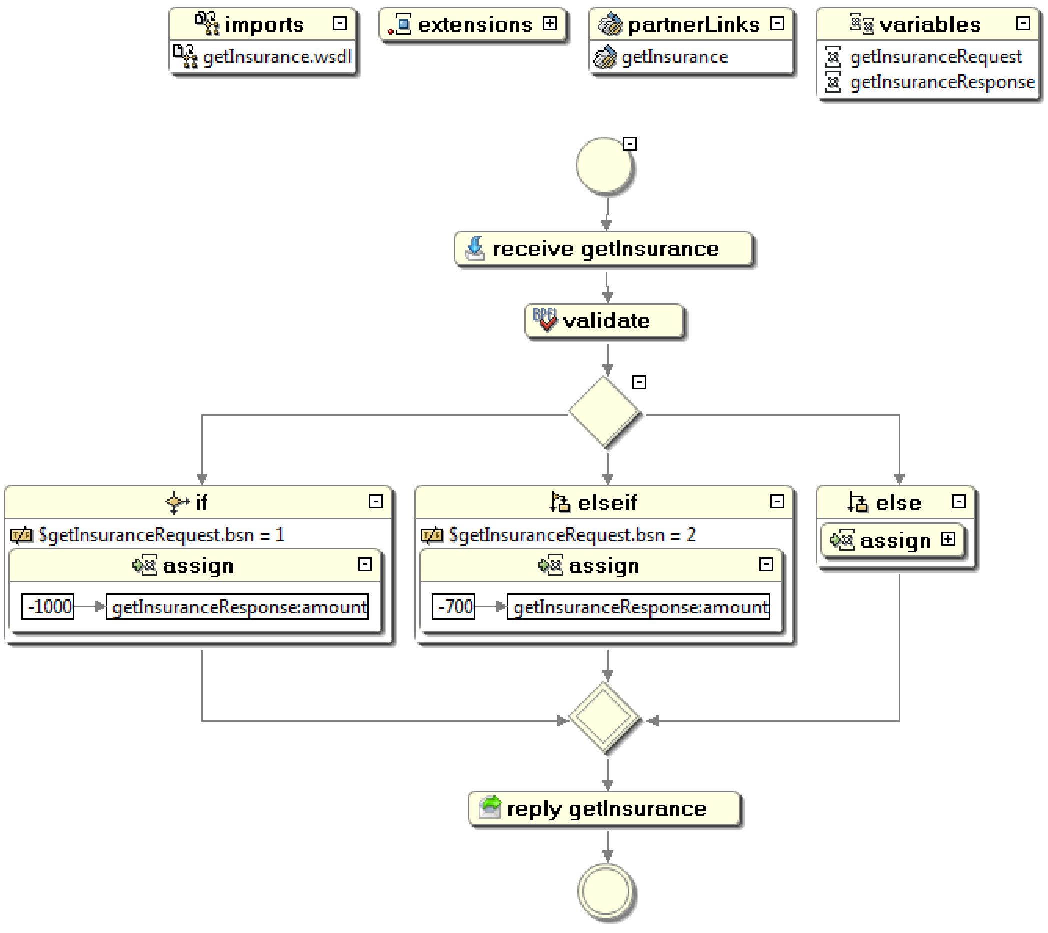
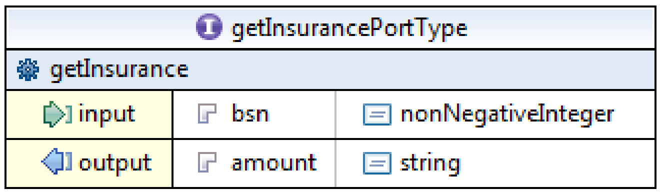


Figure 13: Insurance company simulation in BPEL.

The graphical description of the service's WSDL:



getInsurancePortType		
getInsurance		
input	bsn	nonNegativeInteger
output	amount	string

Figure 14: The graphical description of the service's WSDL.

BSN: The social security number of a person.
AMOUNT: The amount of money a person is insured for.

BSN	AMOUNT
1	1000
2	700
Other than '1' or '2'	0

The getInsurance BPEL simulation test the input argument's format of the service calls. Depending on the BSN the service returns an amount of money.

8.8. *Order supplier BPEL simulation*

The BPEL simulation for an order supplier is called “supplyOrder”. Depending on the BSN number and the request care it gives a response containing a description of the care delivery and the price.

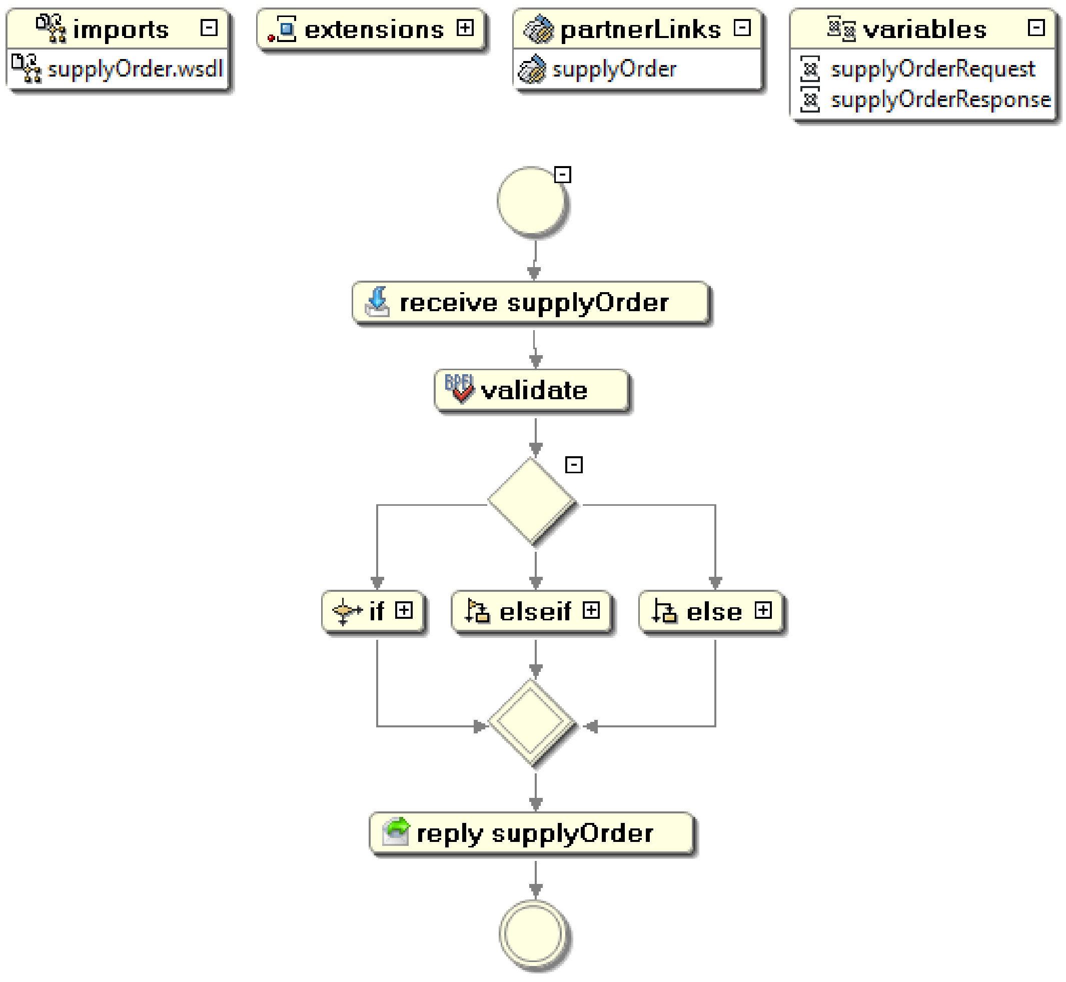
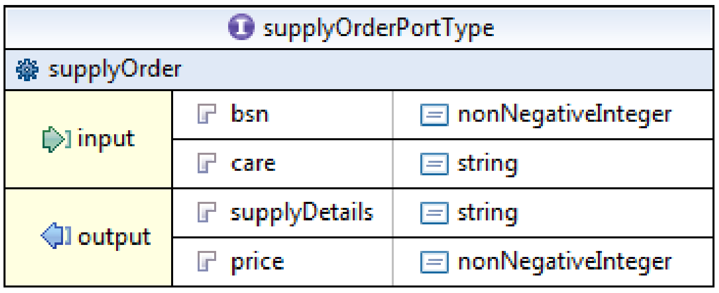


Figure 15: Simulation of supply order service in BPEL.

Graphical description of the supplyOrder WSDL:



supplyOrderPortType		
supplyOrder		
input	bsn	nonNegativeInteger
	care	string
output	supplyDetails	string
	price	nonNegativeInteger

Figure 16: The graphical description of the service's WSDL.

BSN: Person's social security number.
CARE: The type of care that needs to be supplied.
SUPPLYDETAILS: The details of the care supply.
PRICE: The cost of the supply of the care.

The following table described the service's behaviour depending on the input received:

BSN	CARE	SUPPLYDETAILS	PRICE
1	wheelchair	yes	800
1	transportation	yes	800
1	Minor home modification	yes	800
1	Other than above	Type of requested care not supported	0
2	wheelchair	yes	800
2	transportation	yes	800
2	Minor home modification	yes	800
2	Other than above	Type of requested care not supported	0
other	Any string	Client bsn not recognized	0

We can simulate that the care is actually delivered, that the wrong type of care was requested, or that the client bsn is not recognized.

8.9. *Municipality Office receiving WMO-support requests*

All the service in our simulated service-oriented architecture come together in the office receiving actual client's WMO-support requests. When an office receives a WMO-request, it checks the medical advice given by the doctor, checks the amount by which a person is insured, and it tries to order the care requested. After doing all this it reports the result of the WMO-request and the price. The service that represents the office is called “getWMO”.

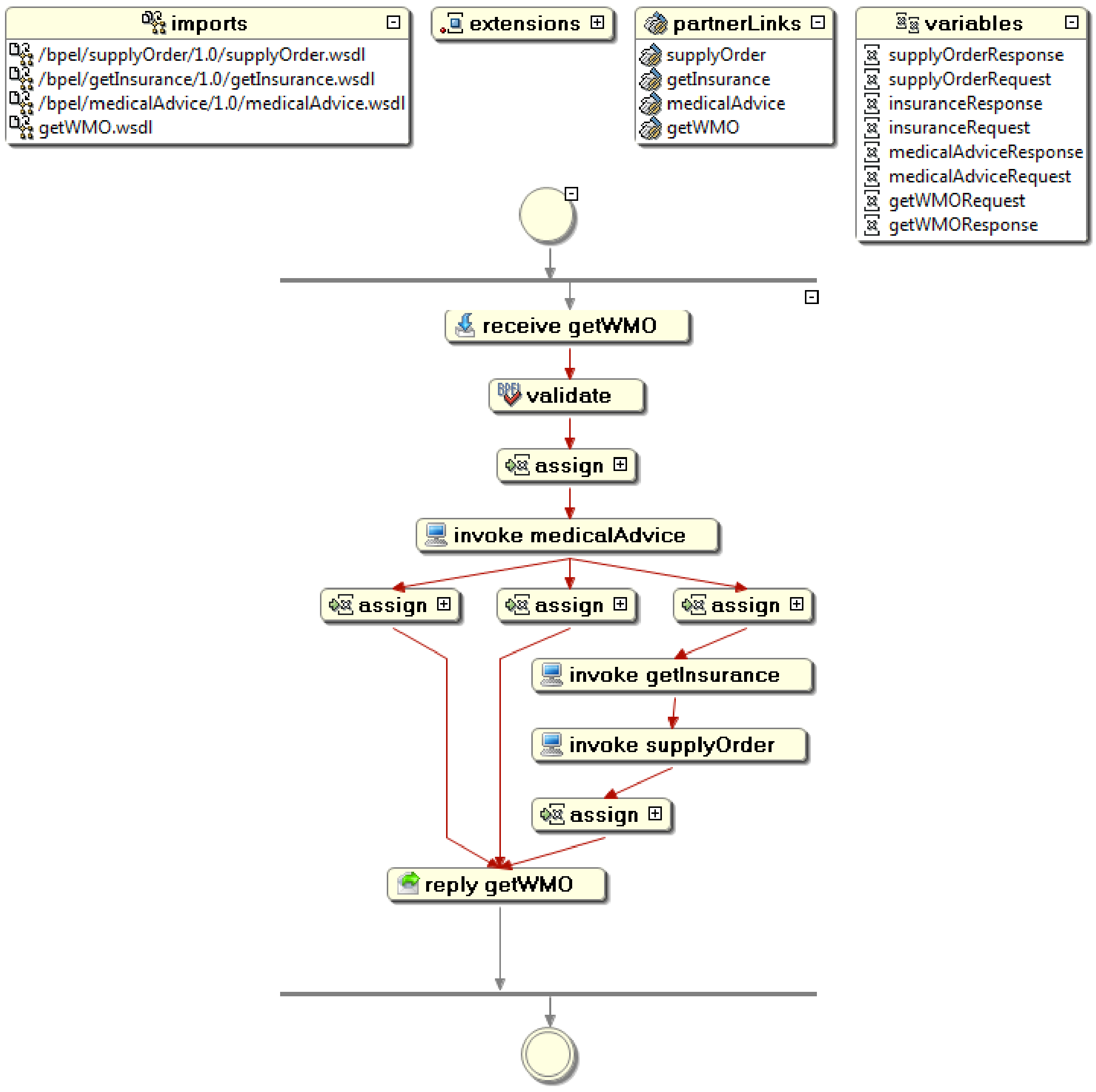
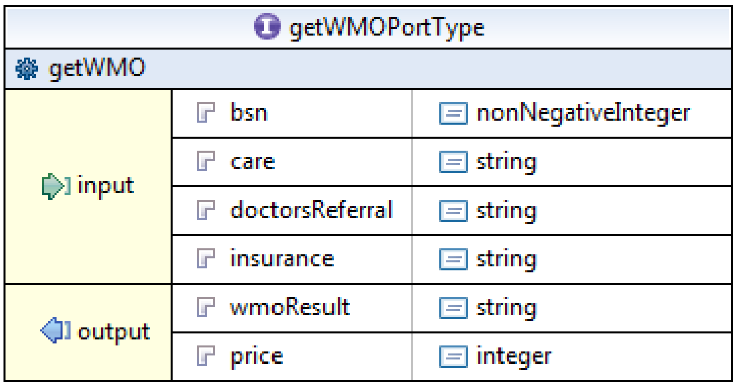


Figure 17: BPEL simulation of WMO SOA.

The graphical description of the getWMO WSDL:



getWMOPortType		
getWMO		
input	bsn	nonNegativeInteger
	care	string
	doctorsReferral	string
	insurance	string
output	wmoResult	string
	price	integer

Figure 18: The graphical description of the getWMO WSDL.

BSN: Person's social security number.
CARE: Type of care requested.
DocterReferral: Does the client have a doctors referral? This can be used by the office employees as extra information, for instance not to make unnecessary service calls.
INSURANCE: Does the person have insurance? This can be used by the office employees as extra information, for instance not to make unnecessary service calls.
wmoResult: What was the result of the WMO-request?
PRICE: The cost of the request after deducting insurance money. If this amount is zero or less than all the care is covered by insurance, otherwise part has to be paid by the client.

The following table describes the behaviour of the “getWMO” BPEL simulation:

BSN	**Care**	**Advice**	**insurance**	**wmoResult**	**price**
1	transportation	yes	Any string	Order will be delivered	-200
1	aspirine	yes	Any string	Type of requested care not supported	0
1	Other	no	Any string	Negative advice	0
2	wheelchair	yes	Any string	Order will be delivered	100
2	Other than 'wheelchair'	yes	Any string	Negative advice	0
Other than '1' or '2'	Any string	unknown	Any string	Doctors advice unknown because he does not recognize bsn	0

8.10. *Adding External services for data access*

When we allow ourselves to go one step beyond BPEL, we can achieve a much better simulation tool than we have right now. Since we can't perform direct data access, e.g. to databases, we have to hard code all the options of our simulation. This is ofcourse not desirable. What we can do in BPEL is calling web services. Therefore we propose to use some simple web service stubs, that allow for access to test data.

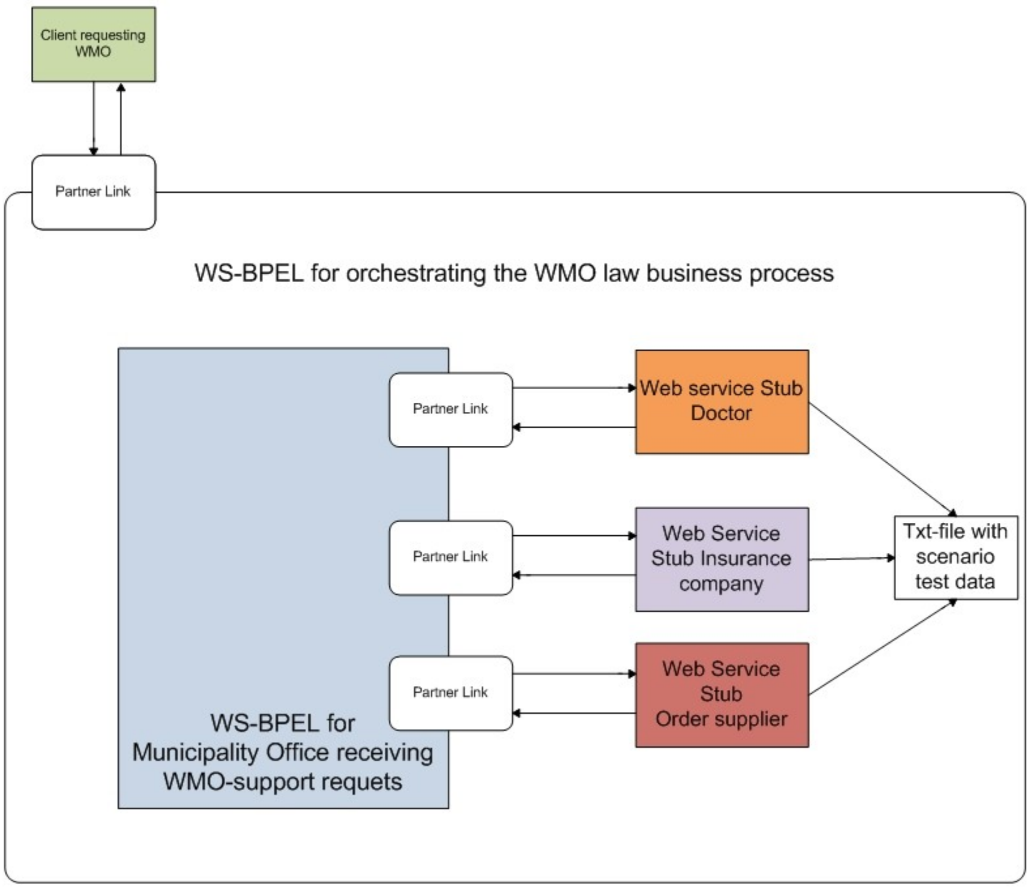

Client requesting WMO
Partner Link
WS-BPEL for orchestrating the WMO law business process
Partner Link
Partner Link
Partner Link
Web service Stub Doctor
Web Service Stub Insurance company
Web Service Stub Order supplier
Txt-file with scenario test data
WS-BPEL for Municipality Office receiving WMO-support requets

Figure 19: The improved solution with java-based web service stubs that can access the test data.

In the previous sections, each invocation in the simulation was made to a web service that was implemented by another BPEL file. In the remaining sections of this chapter we show a solution where each invocation by the simulation is made to a web service stub that can access data. First we will explain the web service stubs, and then we'll show the main simulation BPEL file that orchestrates the service stubs.

8.11. *Java-based web service stubs*

We want to build java stubs that can access data. Since this is a prototype we keep it light and simple, and we will use text files for the data storage.
We will need data for all the components in the WMO simulation: medical advice data, insurance data, and order suplly data. We will seperate the data columns in the text file by the ','-sign. Below is the data file for the examples, of course you could add as many rows as you like. Above the data are the column definitions:

BSN, CARE REQUESTED, MEDICAL ADVICE, INSURANCE($), SUPPLY PRICE

1,	transportation,	yes,	300000,	1000
1,	aspirine,	yes,	300000,	333
2,	wheelchair,	yes,	50000,	3500
3,	valium,	yes,	100000,	100
3,	wheelchair,	no,	100000,	3500

Now we need the stubs to read in the data and respond to any invocations made. The stubs are made in Java.

The **medicalAdvice** web service has an operation called **getAdvice(bsn,care)**, and it returns a string with the advice, meaning whether or not the person really should get the care.

The **getInsurance** web service has an operation called **getAdviceAmount(bsn)**. It returns the amount a person is insured for.

The **supplyOrder** web service has an operation called **getOrderPrice(bsn,care)**. It returns the price of the order if it exists, otherwise it returns -1.

Each of these web services check the data file and respond according to the contents of the data file.

Now we have dynamic web service stubs, and we need the BPEL file to orchestrate them to complete the WMO simulation. The Java-based web service stubs were developed in Open ESB, and we will do the same for our final BPEL simulation, to keep the number of running programs to a minimum.

Below you see the BPEL simulation, together with the web service stubs on the right.

We can now use as much test data as we like, without further complicating the BPEL simulation. We simply edit the text files if we want to add new test scenarios. When someone now wants to test their implementation of a medical advice web service or an insurance web service he or she can simply exchange their application with the corresponding web service stub. Of course their implementation should use the same data set as the stub that was removed.

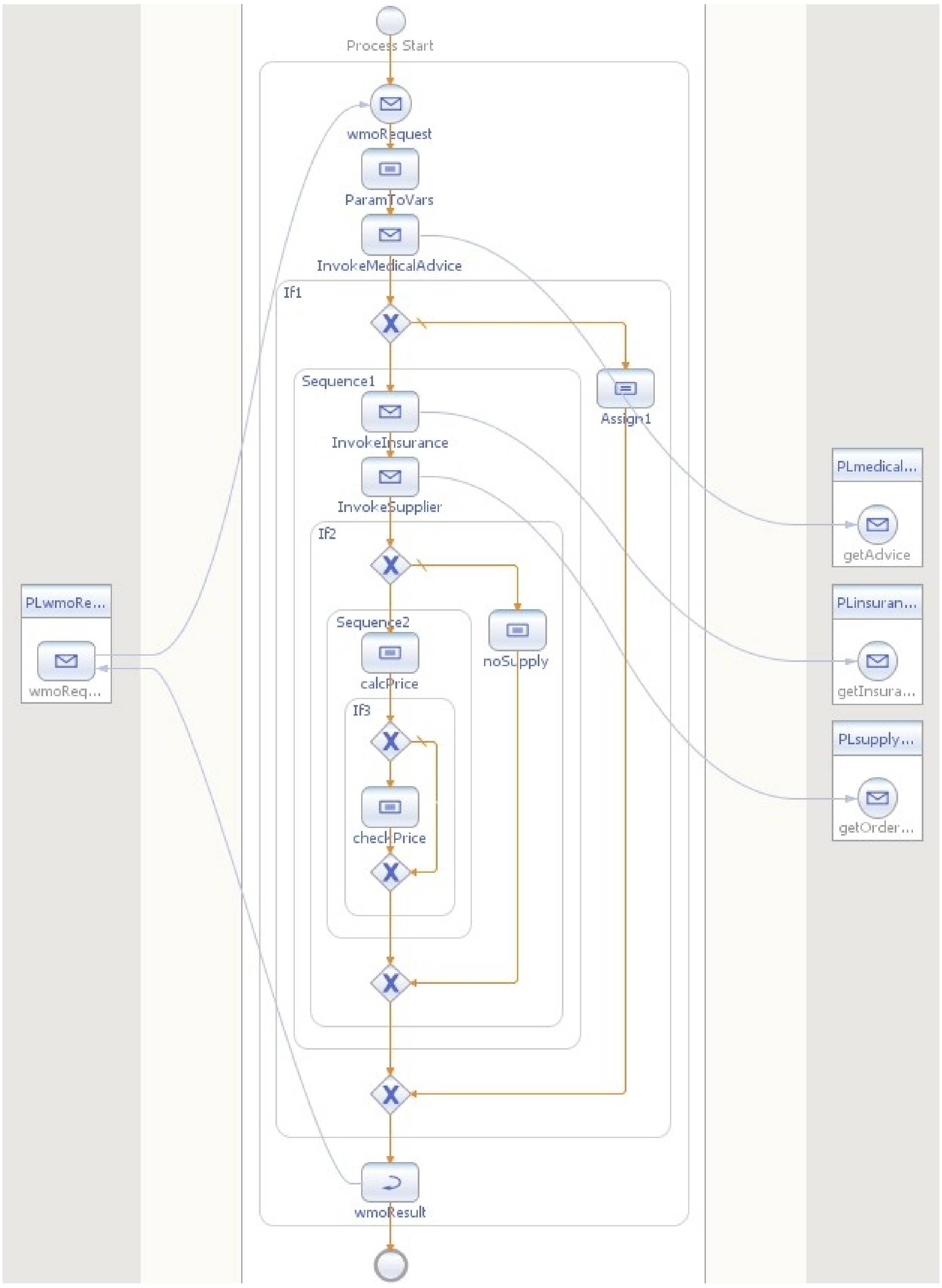


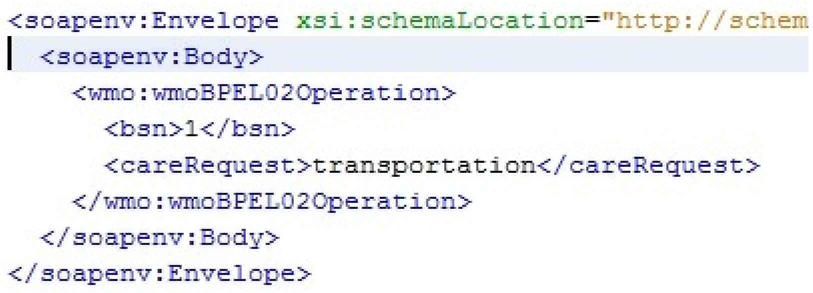
Figure 20: The complete BPEL WMO simulation in Open ESB together with on the right the web service stubs that can access data.

8.12. *Example Scenarios*

Using our new solution we can execute test scenarios. The possibilities are endless of course, but to give an idea we'll show three scenarios that we used with our demo of the solution:

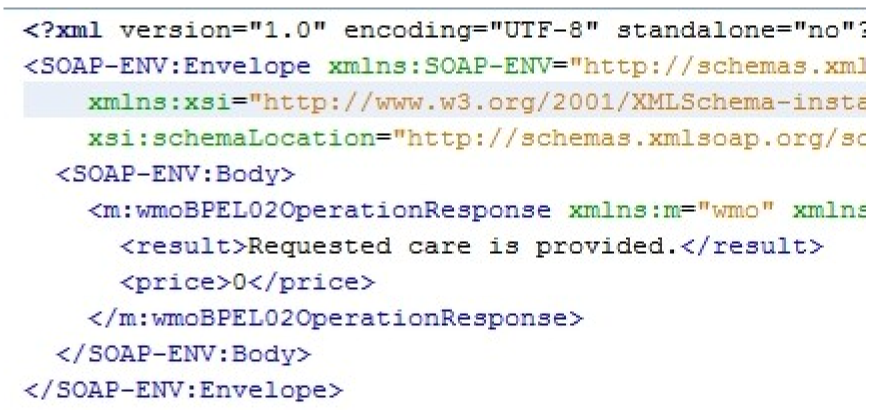
Scenario 1: Successful request and cost covered by insurance.

Request:



```
<soapenv:Envelope xsi:schemaLocation="http://schem
  <soapenv:Body>
    <wmo:wmoBPEL02Operation>
      <bsn>1</bsn>
      <careRequest>transportation</careRequest>
    </wmo:wmoBPEL02Operation>
  </soapenv:Body>
</soapenv:Envelope>
```

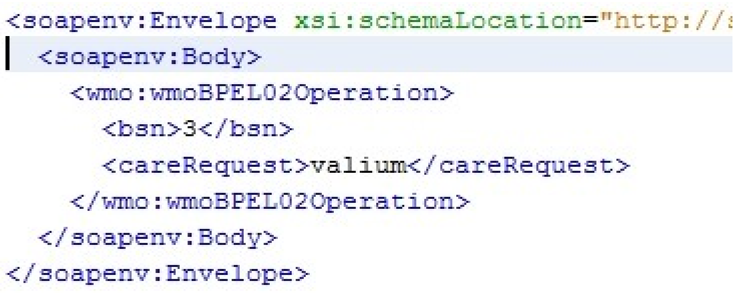
Response:



```
<?xml version="1.0" encoding="UTF-8" standalone="no"?
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xml
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-insta
    xsi:schemaLocation="http://schemas.xmlsoap.org/sc
  <SOAP-ENV:Body>
    <m:wmoBPEL02OperationResponse xmlns:m="wmo" xmlns
      <result>Requested care is provided.</result>
      <price>0</price>
    </m:wmoBPEL02OperationResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

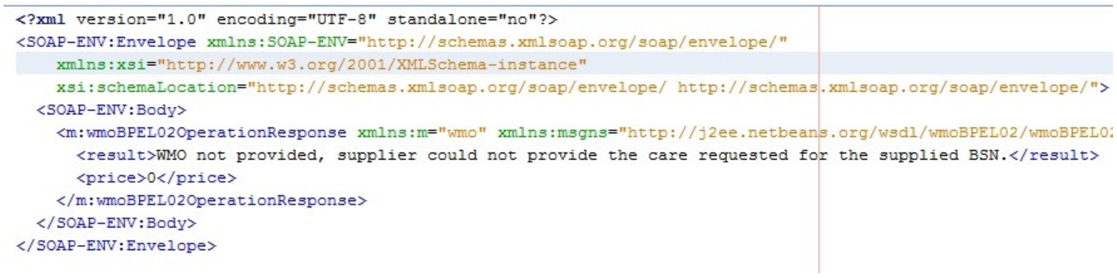
Scenario 2: WMO denied because care could not be supplied.

Request:



```
<soapenv:Envelope xsi:schemaLocation="http://
  <soapenv:Body>
    <wmo:wmoBPEL02Operation>
      <bsn>3</bsn>
      <careRequest>valium</careRequest>
    </wmo:wmoBPEL02Operation>
  </soapenv:Body>
</soapenv:Envelope>
```

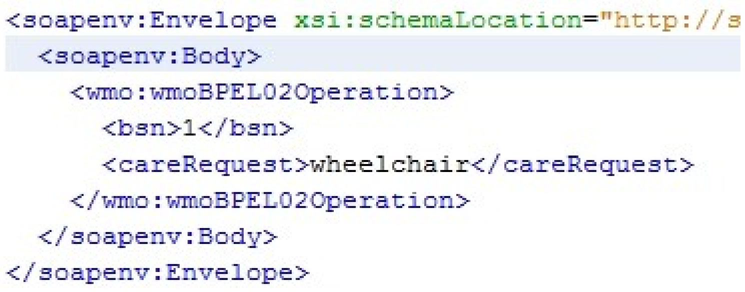
Response:



```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/ http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <m:wmoBPEL02OperationResponse xmlns:m="wmo" xmlns:msgns="http://j2ee.netbeans.org/wsdl/wmoBPEL02/wmoBPEL02
      <result>WMO not provided, supplier could not provide the care requested for the supplied BSN.</result>
      <price>0</price>
    </m:wmoBPEL02OperationResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

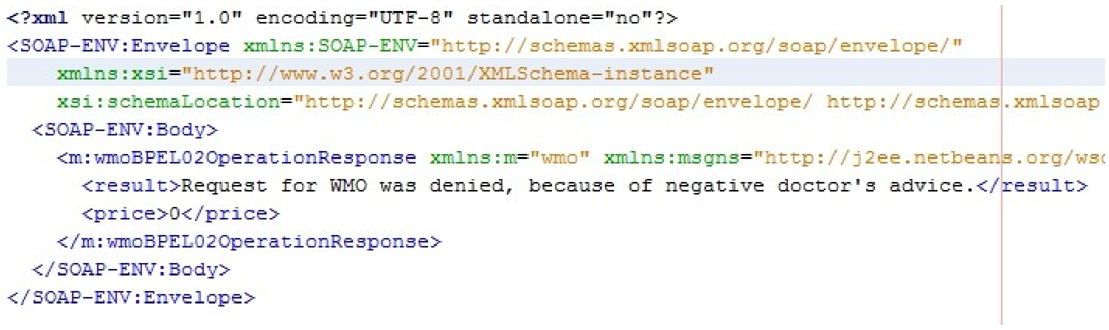
Scenario 3: WMO denied because of negative medical advice.

Request:



```
<soapenv:Envelope xsi:schemaLocation="http://s
  <soapenv:Body>
    <wmo:wmoBPEL02Operation>
      <bsn>1</bsn>
      <careRequest>wheelchair</careRequest>
    </wmo:wmoBPEL02Operation>
  </soapenv:Body>
</soapenv:Envelope>
```

Response:



```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/ http://schemas.xmlsoap
  <SOAP-ENV:Body>
    <m:wmoBPEL02OperationResponse xmlns:m="wmo" xmlns:msgns="http://j2ee.netbeans.org/ws
      <result>Request for WMO was denied, because of negative doctor's advice.</result>
      <price>0</price>
    </m:wmoBPEL02OperationResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Now when we test our implementation of a certain component using the simulation, we simply run the scenario's, and see if everything behaves as expected.

9. Evaluation

In this section we discuss the result of the BPEL-simulation of the WMO service-oriented system we created. Both the implementation of the BPEL-simulation and the details of the service-oriented WMO system are described in the previous chapters.
We will discuss how well we can perform different types of testing using our solution. We will start with the evaluation of non-functional testing. After that we will discuss interface testing, scenario testing, the complexity of the implementation, and we will discuss test coverage.

9.1. *Non-functional testing*

Our BPEL-simulation is more suited for some types of non-functional testing than others. We evaluate the non-functional testing capabilities of our solution here that stand out.

- **Compatibility Testing and integration testing**: Our solution is very well suited for compatibility testing. It can be used for testing a component's compatibility with the rest of a service-oriented system with respect to the web service interface defined in the WSDL'sof the system components. We will get back to this in the section below on *interface testing*. One might also consider this part of integration testing , but integration testing also involves functional aspects, testing whether the system still functions correctly after integration of seperate components. We will discuss more on functional testing with our solution in the section on *functional testing of scenarios* below.
- **Load testing or performance testing**: Although it would be possible to create some sort of BPEL process that would call a service that is to be tested repeatedly, this would make little sense. The performance limit of the BPEL-simulation doing the webservice invocation might be worse than the performance of the tested service for receiving invocations. It would be better to use a special performance testing tool for this (e.g. Load test or SOAtest).

9.2. *Interface testing*

WS-BPEL is very well suited for testing argument formats of web service invocation. Ws-BPEL has a special *validate-tag* defined for this, that checks the format of the arguments used in a web service invocation against the paramater data type definition. For instance when an argument should be an integer, the validate-tag ensures that the argument used in a web service call is indeed an integer, otherwise the WS-BPEL file representing the web service will throw an exception.
We can define extra properties for variables that aren't already definined using XSD (XML Schema Definition) [37], but most well known primitive data types, such as boolean, string, float, et cetera, are already prefefined.
For each parameter we can provide either a correct or an incorrect data type, when we perform an invocation. When we define the number of parameters for a web service invocation as **n**, we could test all combinations of correct and incorrect parameter data type values in $\mathbf{O(2^n)}$ test invocations.

If we think testing a combination of faulty paramaters doesn't add any value with respect to testing the faulty parameters seperately, we could just call a web service once for each parameter, calling with one incorrect type parameter, while we use correct data types for the other paramaters. In that case we could suffice with **O(n)** calls. Either way, it would be interesting to look at automated test data generation for this. We will get back to that in the section on future work. Summarizing this conclusion:

- The proposed solution can be used for interface testing. If we wish to test all possible combinations of a web service component with **n** parameters, this would take **O(2^n)** test calls. If we are satisfied with just testing for each parameter one test call with a correct parameter value and one test call for a parameter value of an incorrect data type, we can achieve this in **O(n)** test calls.

9.3. *Functional testing of scenarios*

We were able to hardcode an example of a positive test case, a negative test case, and a undecisive test case (e.g. WMO support request for person with an unknown bsn number) for each scenario we looked at. We hardcoded the examples using simple statements (e.g. if-else-statements). For instance, we hardcoded the GGD component BPEL simulation (The doctors that give medical advice) to give a positive medical advice for people that either “have a BSN (Dutch social security number) of 1 and request for transportation or aspirine” or for people that “have a BSN of 2 and request a wheelchair”. People with a BSN of 1 or 2 that request any other care than defined in the previous sentence get a negative medical advice, and people with any other BSN than 1 or 2 are considered unknown.

Working like this allows us to provide the testers with data for executing test scenarios for positive medical advice (positive test case), negative medical advice (negative test case), and person unknown (undecisive/negative test case). We repeat the input and output table here from the implementation section for clarity:

BSN	**Care**	**Advice**
1	transportation	yes
1	aspirine	yes
1	Other than 'transportation' or 'aspirine'	no
2	wheelchair	yes
2	Other than 'wheelchair'	yes
Other than '1' or '2'	Any string	unknown

The above data in the table provides a tester with data he can use to see how his or her application reacts to the expected outcome. This type of hardcoding of scenarios can be done for as much scenarios as you want, but it can become quite complicated. So how complex does this actually become? We have found (experimentally from our case study), that the number of branches (if-, elseif-, or else-branches) needed is proportional to the number of scenarios you want to implement. This means:

- For **n** test scenarios we need **O(n)** branches in our hardcoded WS-BPEL simulation.

When we add web service stubs that provide us with data, like we introduced in section 8.10, then we don't need to hardcode test scenarios anymore. Even though this is not a hundred percent BPEL anymore, it is of course a much better solution.

- When we introduce web service stubs for data access, our BPEL simulation doesn't need any hardcoding anymore for covering **n** test scenarios.

9.4. *Simulation complexity*

So how complex will our simulation be with respect to the original simulation? Unfortunately, the simulation will have to have as much BPEL files implementing service components, as the original service-oriented system has web services. This means that for very complex systems it might not be feasable to create a simulation using our proposed solution, especially if the system will change a lot over time.

- For simulating a service-oriented architecture with n services, we need **O(n)** services (BPEL files or web service stubs) in our simulation.

9.5. *Test coverage*

So how do we do when we want to use the proposed solution to achieve a 100% test coverage? To evaluate this, first let us assume the following:

1) We say that we have a 100% test coverage when each feature is tested by at least one scenario.
2) A feature is usually tested using at most **O(1)** scenarios (e.g. scenarios for a succesful test case, and a test case for exceptions like unknown data or incorrect data format).

Using these facts we can derive that:

- For simulating a service-oriented architecture with n web service components, and 100% of the feature testing being covered by m test scenarios, we can achieve a 100% test coverage using the proposed solution with **O(n)** components (WS-BPEL files or web service stubs).

9.6. *Overall decrease of integration testing costs?*

Now that we've shown how we can use a simulated environment for performing “offline” integration testing to decrease the amount of real integration testing that needs to be done, the question arises of how much can we actually save by this on the cost of integration testing? This question is hard to answer using the work in this paper. We propose some experiments in the future work section, that could be done in combination with our work, but are beyond the scope of this project with respect to time and assets. However, we can derive some conclusions from other work, since it shows some resemblance with our work. *Mani, Sinha, Sinha, Dhoolia, Mukherjee, and Chakraborty* claim in their work that they can reduce the amount of test cases that have to be performed on the live service-oriented system by 31% [3]. Since they use an approach that is partly similar to ours, we can expect our simulation to perform alike. In fact, our solution adds functional scenario execution to the simulation, while *Mani, Sinha, Sinha, Dhoolia, Mukherjee, and Chakraborty* only use mock objects with pre- and postcondition and data format tests. It might be possible that the added functionality of our solution could perform even better. Of course, while possible, this is based on other research, and we can't tell for sure that it will work. We will discuss in future work how we could set up an experiment to get more definitive numbers. But for now we can say:

- Other research suggests that we can decrease the amount of test cases that have to be performed on the actual running service-oriented service by at least 30%. although this is by no means proof, we think there is a strong indication that this can be achieved.

10. Future work

10.1. *Automation of BPEL files*

A good addition to this work would be to automate the generation of the simulation in WS-BPEL and the required web service stubs. The WS-BPEL file and web service stubs could be generated from the original WSDL files of the service-oriented architecture. This could contribute a lot to the usefullnes and usability of the solution.

10.2. *Automation of test data generation*

In the section on evaluation we also mentioned how interface testing can be achieved with the proposed solution. It turned out that this could be achieved in doing **O(2^n)** test calls, or in O(n) if we would do it less exhaustively. Since these test calls involve simple and predictable calls, derivable from the WSDL-files, we think it would be very well possible to automate the test calls and the argument values to go with it. Future research could try to show that this is possible and create some sort of implementation of it along the way.

10.3. *Further experiments*

To give a stronger argumentation for using the proposed solution or some sort extended or improved version of it, it would be nice to have more numbers to support the theory. We propose an experiment set up that could be used in future work. For the scope of this work it is unfeasable to set up such an experiment, that's why we propose it in this section.
We propose to find a simple service-oriented architecture that is up and running, and accessible throught the internet. We build a simulation of this service-oriented architecture using our proposed solution. Now two groups of programmers are needed. Both groups of programmers implement a component that has to be hooked up to the service-oriented architecture. They get the same documentation, but only one group gets the instruction to use the simulation and use it for testing, until they find no more errors with it before going to live testing. The other group should directly go on to live testing, without using th simulation at all. It will be interesting to see the testing time and amount of tests that will be saved by the group using our solution. We might also find less expected side effects. For instance, the group of programmers that perform direct live testing might turn out to program more precisely, going over their work more often before testing, because they know unsuccesful tests will bring extra costs.

11. Conclusion

When connecting a new component to a service-oriented system, integration testing has to be performed. A service-oriented system can't be taken offline easily for testing since system components are distributed over different locations, and different components are owned by different parties. This means that a service-oriented system often has to keep running during integration testing. It may become polluted with test data, and system performance can decrease by the load of test calls. Also, if the service-oriented system uses paid services, test calls cost extra money.

In this paper we have created a simulation of a service-oriented architecture. The idea is when a new service component needs to be connected to the service-oriented architecture, we first perform as much of the integration testing as possible on the simulation. By doing this we try to find as many errors as possible offline before proceeding to performing integration testing onto the actual service-oriented system. This way we minimize the burdon of integration testing on a running service-oriented system.

The simulation we made was made in WS-BPEL. We chose to use WS-BPEL because we suspected it was well suited because of the service-oriented nature of the language. We wanted to evaluate how well suited WS-BPEL would be for the job.

We managed to create the simulation using WS-BPEL. For each web service in the simulation we also managed to hardcode some functionality that simulate some system functionality. By doing that we created the possibility of executing test scenarios using the simulation. We also created an alternative version which works better, but uses web service stubs that can access data, and are not pure BPEL anymore.

During our evaluation the simulation of a service-oriented architecture we created in WS-BPEL, we found out the following:

- We can test a web service components interface that has **n** parameters with **$O(2^n)$** test calls. If we are less precise we could do it in **$O(n)$** test calls. This shows us that our solution can be used for interface compatibility testing, but also that automatic test data generation is an interesting option for future work.
- We can hardcode **n** testing scenarios using **$O(n)$** branches in our simulation. When we extend our solution with web service stubs that can access test data we don't have to hardcode testing scenarios anymore, and the solution becomes suited for any number of testing scenarios.
- If we have a service-oriented architecture consisting of **n** web service components, and requiring the execution of **m** test scenarios for a 100% test coverage, we can achieve a 100% test coverage with a simulation consisting of **$O(n)$** components (WS-BPEL files or service stubs). This means we can achieve 100% test coverage for any number of test scenarios with our simulation.
- We have found other research suggesting that testing using a simulation can achieve a decrease of 31% in tests that have to be executed on the actual running service-oriented system.
- Given the indications we have found that our solution would be better suited for smaller less complicated service-oriented architectures, we think it would be well worth to do further research for finding ways to automation of creating

our solution. Using test data and expected responses to those test data together with the actual system's WSDL files it may be able to fully automate our solution of creating a simulation. This would make our solution better suited for large and complicated systems.

- Another good subject for future work would be experiments that would further validate the benefits of our proposed method of using a WS-BPEL simulation of a service-oriented architecture for performing offline integration testing.

Alltogether we think that WS-BPEL is suited for the job of simulating service-oriented architectures, and its use can contribute to the decrease of integration testing costs for new components on running service-oriented architectures. We have also found out that it is very important to have some sort of access to a database or text files with test data. Since this is not possible with pure BPEL we think it is necessary to extend the BPEL solution with web service stubs that can access data. In our case we used Java for these stubs, but it doesn't really matter what kind of language is behind these services, as long as they are web services that can be used in a BPEL implementation. We also think that extensive further research is needed to further validate the method, and to improve and if possible even automate it.

12. References

[1] Provided by dr. Alexander Lazovik, assistant professor at the Rijksuniversiteit Groningen.

[2] *Gerardo Canfora and Massimiliano Di Penta*. **"Testing services and service-centric systems: Challenges and oppurtunities."**. IT Pro, 2006.

[3] *Mani,sinha,Sinha,Dhoolia,Mukherjee,Chakraborty.* **"Efficient Testing of Service-Oriented Applications Using Semantic Service Stubs"**.

[4] *Xiaoying Bai, Wenli Dong, Wei-Tek Tsai, Yinong Chen.* **"WSDL-Based Automatic Test Case Generation for Web Services Testing".** In *Proc. of the Intl. Workshop on Service-Oriented Syst. Eng.*, pages 207–212, Oct. 2005.

[5] *Nikolai Tillmann, Jonathan de Halleu.* **"White-Box Testing of Behavioral Web Service Contracts with Pex"**. International Symposium on Software Testing and Analysis 2008.

[6] *Raquel Blanco, José García-Fanjul, Javier Tuya.* "**A first approach to test case generation for BPEL compositions of web services using Scatter Search**". International Conference on Software Testing, Verification, and Validation 2009.

[7] *ChangSup Keum, Sungwon Kang, In-Young Ko.* "**Generating Test Cases for Web Services Using Extended Finite State Machine**". Proc. of the 16th IFIP Intl. Conf. on Testing Communicating Systems, pages 103–117, May 2006.

[8] *Sylvain Halle, Graham Hughes, Tevfik Bultan, and Muath Alkhalaf;* "**Generating Interface Grammars from WSDL for Automated Verification of Web Services**"; International conference on service oriented computing 2009, icsoc09.

[9] Tsai, W.T. , Wei, X. , Chen, Y. , Paul, R; "**A Robust Testing Framework for Verifying Web Services by Completeness and Consistency Analysis**"; Service-Oriented System Engineering, 2005. SOSE 2005. IEEE International Workshop.

[10] H. M. Sneed and S. Huang; **"WSDLTest – A Tool for Testing Web Services"**; In *Proc. of the Intl. Symp. on Web Site Evolution*, pages 14–21, Sept. 2006.

[11] *Evan Martin, Suranjana Basu, Tao Xie;* "**Automated Testing and Response Analysis of Web Services**"; Conference on Web Services, 2007. ICWS 2007.

[12] *Xiaoying Bai, Wenli Dong, Wei-Tek Tsai, Yinong Chen;* "**WSDL-Based Automatic Test Case Generation for Web Services Testing**"; Service-Oriented System Engineering, 2005. SOSE 2005. IEEE International Workshop.

[13] *Cesare Bartolini, Antonia Bertolino, Eda Marchetti and Andrea Polini;* "**Towards Automated WSDL-Based Testing of Web Services**"; Service-Oriented Computing, ICSOC 2008.

[14] *Amit Paradkar and Avik Sinha;* "**Specify Once Test Everywhere: Analyzing Invariants to Augment Service Descriptions for Automated Test Generation**"; Service-Oriented Computing, ICSOC 2008.

[15] H. Huang, W.-T. Tsai, R. Paul, and Y. Chen. "**Automated Model Checking and Testing for Composite Web Services**" In *Proc. of the 8th Intl. Symp. on Object-Oriented Real-Time Distributed Computing*, pages 300–307, May 2005.

[16] *Becker, K., Lopes, A., Milojicic, D.S., Pruyne, J., Singhal, S.;* "**Automatically Determining Compatibility of Evolving Services**"; IEEE International Conference on Web Services, 2008. ICWS '08.

[17] *Reiko Heckel and Leonardo Mariani;* "**Automatic Conformance Testing of Web Services**"; Fundamental Approaches to Software Engineering. Lecture Notes in Computer Science, 2005.

[18] *Guilan Dai, Xiaoying Bai, Yongbo Wang, Fengjun Dai;* **"Contract-Based Testing for Web Services"**; 31st Annual International Computer Software and Applications Conference, 2007.

[19] *Reiko Heckel, Marc Lohmann;* **"Towards Contract-based Testing of Web Services"**; Electronic Notes in Theoretical Computer Science (ENTCS), January 2005.

[20] *Tsai, W.T.; Paul, R.; Yamin Wang; Chun Fan; Dong Wang;* **"Extending WSDL to Facilitate Web Services Testing"**; 7th IEEE International Symposium on High Assurance Systems Engineering, 2002.

[21] *Li Li, Wu Chou;* **"An Abstract GFSM Model for Optimal and Incremental Conformance Testing of Web Services";** IEEE International Conference on Web Services, 2009. ICWS 2009.

[22] *Tim Mackinnon, Steve Freeman, Philip Craig;* **"Endo-Testing: Unit Testing with Mock Objects";** eXtreme Programming and Flexible Processes in Software Engineering – XP2000;

[23] *Tillmann, N.; Schulte, W.;* **"Mock-object generation with behavior"**; 21st IEEE/ACM International Conference on Automated Software Engineering, 2006. ASE '06.

[24] *Masahiro Tanaka and Toru Ishida;* **"Predicting and Learning Executability of Composite Web Services";** International conference on service oriented computing.

[25] *Carsten Hentrich, Uwe Zdun;* **"Patterns for Process-Oriented Integration in Service-Oriented Architectures"**;
http://www.infosys.tuwien.ac.at/staff/zdun/publications/eplop06.pdf

[26] S*ystem and software engineering and distrbuted systems group, Rijksuniversiteit Groningen;* **"Stessi research Proposal"**; Internal R.U.G. Document, 2009.

[27] *Bertolino, A.; Polini, A.;* **"The Audition Framework for TestingWeb Services Interoperability"**; 31st EUROMICRO Conference on Software Engineering and Advanced Applications, 2005.

[28] *Peter Budny, Srihari Govindharaj and Karsten Schwan;* **"WorldTravel: A Testbed for Service-Oriented Applications"**; International conference on Service-Oriented Computing, ICSOC 2008.

[29] http://www.testdata.nl/index.php?option=com_content&view=article&id=71:cursus-burgerzaken-leidt-tot-geboortes&catid=39:knowledgebase&Itemid=60

[30] D. Booth et al. **"Web Service Architecture"** http://www.w3.org/TR/ws-arch/ 2004.

[31] "Web Service" http://en.wikipedia.org/wiki/Web_service.

[32] *Kodali, R.R.* "**What is service-oriented architecture?**" JavaWorld.com (June 2005).

[33] "**Service-oriented architecture**" http://en.wikipedia.org/wiki/Service_Oriented_Architecture.

[34] *Jacobson, I.,* "**The Object Advantage**" 1995 ISBN 0 201 42289 1

[35] *Ilia Bider.* "**State-Oriented Business Process Modeling: Principles, Theory and Practice",** 2002.

[36] **"Business process"**. http://en.wikipedia.org/wiki/Business_process.

[37] *Alves et al.* **Web Services Business Process Execution Language Version 2.0**

[38] *Panagiotis Louridas.* **"Orchestrating Web Services with BPEL,"** IEEE Software, vol. 25, no. 2, pp. 85-87, March/April, 2008.

[39] *Sommerville.* "**Software Engineering 7th edition**" Addison-Wesley 2004.