A Study of Web Service Adaptation at Runtime

Simeon Reilly
B.A.(Mod.) Computer Science
Final Year Project April 2011
Supervisor: Dr. Siobhán Clarke

School of Computer Science and Statistics
O’Reilly Institute, Trinity College, Dublin 2, Ireland
DECLARATION

I hereby declare that this project is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university

Name – Simeon Reilly

Date - 11th April 2011
Abstract

We cannot always account for all the possible scenarios a program may encounter during its execution. It is often desirable to make changes to a program while it is still running. This project is a study of the methodology that is used to make web services adaptable at runtime. An evaluation of state of the art approaches is described, and two templates for adaptation are introduced.
Acknowledgments

I would like to thank my project supervisor, Dr. Siobhán Clarke for her guidance and support throughout the project along with research fellow, Razvan Popescu for technical guidance. Also, thanks to my good friends Stephen McEvoy, and Odhran Gavin for the use of some necessary hardware.
Table Of Contents

Chapter 1: Introduction ................................................................................................ 6
  1.1 Motivations ............................................................................................................. 6
  1.2 Project Goals ........................................................................................................... 7
  1.3 Guide to Report ..................................................................................................... 7

Chapter 2: Background Information ............................................................................. 8
  2.1 Web Service ............................................................................................................ 8
  2.2 Business Process Execution Language ................................................................. 9

Chapter 3: Approaches ............................................................................................... 12
  3.1 TailorBPEL ............................................................................................................. 12
  3.2 FlexBPEL ................................................................................................................ 13
  3.2 TRAP/BPEL ............................................................................................................ 15
  3.3 Dynamic Linking as an Approach .......................................................................... 15
  3.4 Dino Project .......................................................................................................... 17
  3.5 Apache ODE Extensions ........................................................................................ 18

Chapter 4: Implementation and Evaluation ............................................................... 21
  4.1 Approach Assessment .......................................................................................... 21
    4.1.2 TRAP/BPEL ..................................................................................................... 21
    4.1.3 FlexBPEL ......................................................................................................... 22
    4.1.4 Dino Project ................................................................................................... 23
    4.1.4 Manageable and Adaptable Service Compositions ....................................... 23
    4.1.5 Apache ODE Modification .............................................................................. 24
  4.2 Dynamic Linking Implementations ....................................................................... 25
    4.2.1 Dynamic Configuration .................................................................................. 25
    4.2.3 Dynamic Adaptation with Fault Handling ...................................................... 28

Chapter 5 .................................................................................................................... 29
  Conclusions ................................................................................................................. 29
  5.1 Future Work .......................................................................................................... 30
  5.2 Knowledge Gained ................................................................................................ 31

References .................................................................................................................. 32
Chapter 1

Introduction

This project is a study of approaches for introducing adaptations to web services at runtime, with a particular focus on web service assemblies. Within the report an adaptation is regarded as a modification to a software system that can be defined by a user or the system itself, often as a reaction to some event.

1.1 Motivations

The reasons we desire adaptations in software are threefold.

1. To extend the software with new behaviour, while retaining old behaviour
2. To react to a change in the software’s requirements.
3. To increase the robustness of the software.

Changing a program’s behaviour during design-time is a well understood procedure. However there are situations in which we cannot allow for a program to go offline. These are generally of the form of applications that have multiple users interacting continuously and arbitrarily, or where system interactions are of high importance, or a combination of both.

One such application is crisis-management software for cities.
Such software includes a wide selection of embedded devices which as a unit react to emergencies in urban environments. The Distributed Services Group in TCD has identified a major requirement of this software is the need for wide scale adaptation [1].

1.2 Project Goals

This paper explores a variety of solutions that have emerged in the past ten years to implement adaptable behaviour in web services. State of the art solutions are examined, compared, and evaluated. Finally a practical implementation of select techniques is explored in two software templates.

1.3 Guide to Report

Chapter 2: Background Information

This chapter describes the variety of technologies used.

Chapter 3: Approaches

This chapter discusses the various approaches currently utilized and examines state of the art leading projects

Chapter 4: Implementation & Evaluation

This chapter evaluates approaches discussed in chapter 3, and introduces two frameworks for adaptations.

Chapter 5: Conclusions

Conclusions and future work is discussed, along with knowledge gained.
Chapter 2

Background Information

2.1 Web Service

A web service is defined by the World Wide Web Consortium as,

“...a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format” [2]

This interface is described by a WSDL\(^1\) file. Web Service interfaces are analogous in nature to interfaces commonly used by imperative programming languages. A WSDL file contains the following objects:

<table>
<thead>
<tr>
<th>Namespace</th>
<th>Locations of references for types used in the document</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binding</td>
<td>Specifies the SOAP(^2) binding style for the service.</td>
</tr>
<tr>
<td>Operations</td>
<td>A list of operations (methods) the web service uses</td>
</tr>
<tr>
<td>Messages</td>
<td>Defines the types of the messages used, including input, output, and faults.</td>
</tr>
<tr>
<td>Types</td>
<td>Describes types used in the service. Specified by an internal/external XML Schema</td>
</tr>
<tr>
<td>Interface</td>
<td>Includes the interface name, operations, and messages used by the service.</td>
</tr>
<tr>
<td>Address</td>
<td>The endpoint of a web service. It’s actual connection address.</td>
</tr>
</tbody>
</table>

*Fig. 1. A Table of WSDL Objects*

---

\(^1\) Web Service Definition Language

\(^2\) Simple Object Access Protocol. SOAP is a protocol for exchanging the XML defined information in messages as used by web services.
The two major classifications of web services are those services that are stateful – services that preserve a state between invocations, and stateless – services that do not preserve a state between invocations. Actual web service implementations are diverse. Web services can be written in Java, Python, Perl, C++ and PHP [3]. Web services have proven to be a popular development tool, with support in the popular development environments Eclipse, Netbeans, JDeveloper, and Visual Studio.

2.2 Business Process Execution Language

BPEL is a web technology originally co-designed by IBM and Microsoft in 2003 and subsequently accepted as a standard by OASIS 1 in 2004 [4]. A business process is a selection of related activities that collectively work towards a particular goal. BPEL realizes a business process in the form of orchestrating a set of web services using activity elements.

As can be expected from an executable language these activities take the form of commonly used programming entities, for example <ASSIGN>, <IF>, <WHILE>, <REPEATUNTIL>, <WAIT>, <THROW>, and <SEQUENCE>. An important distinction of BPEL is that it does not alter behaviour of the web services it invokes; rather it provides the orchestration logic between them.

1 The Organization for the Advancement of Structured Information Standards, an international consortium driving web service and e-business standards.
Like the web services they refer to, BPEL files have an interface specified by WSDL, importantly defining the operations and message types within the BPEL process. BPEL utilizes an XML format for its language structure. One of the advantages of the structured, hierarchical nature of the language is that it can be interpreted as a workflow, which development environments, including JDeveloper, Eclipse and Netbeans, strive to take advantage of.

BPEL files are executed by a BPEL engine of which there are several implementations (Apache ODE, Glassfish ESB, Oracle BPEL Process Manager), but which all adhere to the same standard set out by OASIS.
Fig. 2. A sample BPEL Process for a travel booking agency¹

¹ Partner links shown are BPEL entities that encapsulate web services within a BPEL file. Partner links reference WSDL interfaces of the web services utilized.
Chapter 3

Approaches

There has been research into providing adaptable behaviour for BPEL processes since shortly after its adoption, with the first white papers appearing around 2006. A comparison of solutions is illuminative in deriving any framework for adaptive BPEL processes.

3.1 TailorBPEL

TailorBPEL is a German project led by researchers at the University of Bonn. A major design goal is the ability to create a system where even people without technical expertise can modify BPEL processes in an intuitive matter, including at runtime without having any downtime of the core BPEL processes [5]. TailorBPEL utilizes architecture abstraction.

A provider tier represents the providers of web services. A portal tier represents a “cloud service discovery system” to query registries in the provider tier for new or updated web services. An orchestration tier represents an engine to deploy personalized BPEL processes to users. This is a copy of the base BPEL process for the user including extra nodes that combined with other components of the orchestration tier allows modifications to the current running BPEL
process. A portal tier provides logic to bridge end user clients and the orchestration server. It also provides a method to convert BPEL files into human readable vector diagrams intended to allow easy manipulation for end users.

Fig. 3. TailorBPEL Architecture [5]

3.2 FlexBPEL

FlexBPEL is an attempt to address dynamic change problems of BPEL processes [6]. FlexBPEL utilizes a meta-model approach. BPEL
processes have meta-objects associated with them that describe additional structural and semantic information about that process. These meta-objects specify data such as dependencies and behavioural consistency requirements for BPEL process elements. Change to processes is accomplished by using modification primitives on meta-objects. Example primitives include AddLink, AddActivity, and RemoveVariable.

Any changes to meta-objects causes a set of rules, meta-object protocols to fire. These protocols ensure that data in the base model is aligned to data in the meta-model, and vice-versa. Finally a reflection protocol applies changes to process instances based on the information from meta-object protocols.

Fig. 4. A Change Of A BPEL Process Instance [6]
3.3 TRAP/BPEL

TRAP/BPEL is a framework for supplying automotive adaptive behaviour into existing BPEL processes. A major design goal of the TRAP/BPEL framework is to make BPEL processes as robust as possible, emphasizing automotive configuration, optimization, and fault handling [7].

TRAP/BPEL uses generic proxies to achieve its adaptive behaviour. BPEL processes prepared for use with the TRAP/BPEL framework make all web service calls through the use of such proxies. Every proxy has a policy for recovery of web services, including actions to take if said services either fail or operate slowly. There are two recovery actions specified by the proxy recovery policies

1. Retry calling the web service
2. Find a substitute for the web service

3.4 Dynamic Linking\textsuperscript{1} as an Approach

Regular BPEL applications use static partner links. A partner link is the encapsulation of a web service as used in the BPEL language. In static-linking the developer specifies all behaviour relating to a BPEL process at design-time, including the web services with which it converses. However it is within the BPEL standard to use dynamic links. Dynamic linking allows partner links to point to different services.

\textsuperscript{1} Sometimes called dynamic binding
endpoints during runtime, provided that the WSDL interfaces of all service references are equivalent, including the messages, types, and operations that web service utilizes. The actual endpoint address of this implementing service is variable.

Fig. 5. Abstract processes demonstrating static and dynamic linking

3.4.1 Manageable and Adaptable Service Compositions

MASC is a policy based framework that utilizes dynamic links to specify discovery and selection of services, the need for adaptation in processes and process reconfiguration to recover from faults [8]. Policies for web services are included within BPEL processes via XML comments so as to not disturb BPEL compatibility. MASC middleware
then queries a registry of available services based on that policy. New services can be added to the registry allowing new functionality to be introduced during the runtime of a given process.

MASC Middleware queries services that have degraded quality of service. If a web service A has an equivalent web service B in the registry then A and B can be switched, allowing the best performing service to be called by the process. MASC then utilizes a DynamicProxy component as a mediator for dynamically linking the returned web service, completing the web service call.

### 3.5 Dino Project

The Dino project utilizes a different approach for achieving adaptability with regards to web services from those previously described. Dino supports all stages of service composition, including adaptation [9]. Their research suggests that the web service interface specified by WSDL is insufficient, and does not contain enough semantic data, permitting ambiguities in service descriptions. Instead Dino utilizes OWL-S, an ontology that provides additional information about web services including

- service profiles - used to describe the web services functionality,
- process models - used to describe the web services usage particularly inputs and outputs
- service groundings – used to specify technical details for interaction with the service such as port numbers [10]
Within OWL-S all web services are providers, or requestors, or both. Providers have a list of all the capabilities associated with them. Similarly, requestors have a list of all the requirements they need.

Within Dino, service interfaces also include modes of operation that describe different states a service may occupy. Each mode can have different requirements for other services that it requires or provides. The Dino project specifies a Dino broker to mediate these services and provide endpoints to clients and servers. Using the rich level of information provided by the service interfaces, service composition can be achieved without the use of BPEL¹.

Adaptation actions occur when the Dino broker discovers a fault in a service entity. After a set of retries takes place (if appropriate), the broker examines available services with matching capabilities/requirements and supplementary invokes them.

### 3.6 Apache ODE Extensions

One field of thought is that existing BPEL engines aren’t capable of defining adaptation behaviour, at least - not without being modified. A post-graduate in the University of Stuttgart, Mattanja Kern has developed a modification framework to allow modifications to a BPEL

---

¹ Dino does specify that the conventional use of BPEL is possible for specifying service conversations. However it also states that it does need to be modified to take advantage of the Dino infrastructure.
process during its runtime. This framework provides some simple modification operations like inserting or skipping BPEL activities and modifying the value of process variables. The inserted BPEL activities can be provided to the web service in the form of BPEL code as if this fraction of the process has been part of the process in the first place [11]. He allows for this functionality by modifying a popular BPEL engine, the Apache ODE.

An example usage would be to add an activity to an active BPEL process. To achieve this, a SOAP message is sent to the candidate process for modification.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header/>
  <soapenv:Body>
    <mod:InsertNextActivity>
      <ActivityLocation>
        <Process>
          <pid>{http://ode.apache.org/sample/LoanProcess}LoanProcess-1</pid>
          <Name></Name>
          <Namespace></Namespace>
        </Process>
        <Instances>
          <!--Zero or more repetitions:
          <Instance>
            <iid>?</iid>
            <CorrelationProperty>?</CorrelationProperty>
          </Instance>
        -->
        </Instances>
      </ActivityLocation>
      <NewActivity><![CDATA[
        <bpel:assign
          targetNamespace="http://ode.apache.org/sample/LoanProcess"
          xmlns:tns="http://ode.apache.org/sample/LoanProcess"
          xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
          validate="no" name="updateCurrentLoanAmount2">
          <bpel:copy>
            <bpel:from><![CDATA[$ProcessLoan/tns:LoanCurrentAmount - 13]]]><![CDATA[</bpel:from>
            <bpel:to><![CDATA[$ProcessLoan/tns:LoanCurrentAmount]]]><![CDATA[</bpel:to>
          </bpel:copy>
        </bpel:assign>]]></NewActivity>
      <InsertOptions>
        <EventType></EventType>
        <RecoveryAction></RecoveryAction>
      </InsertOptions>
    </mod:InsertNextActivity>
  </soapenv:Body>
</soapenv:Envelope>
```
Figure 6 shows a SOAP message introducing a new assign activity to an active process, “LoanProcess”. All data necessary for the assign activity is included in the message, notably the position of the assign with the process (after the activity payOffReply), the scope of the activity (none), the name of the assign (updateCurrentLoanAmount2), and the copy data itself. Within the modified BPEL engine, a modification compiler parses the new activity and so modifies the original process to reflect the change. The modified engine has similar use cases for the deletion of activities, and the skipping of activities.
Chapter 4

Implementation and Evaluation

4.1 Approach Assessment

A major goal of this project is to evaluate the practicality of solutions described with the current range of software available. A comparison of the state of the art yields the following table.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>TRAP/BPEL</td>
<td>No</td>
<td>Limited</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>FlexBPEL</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Dino Project</td>
<td>n/a</td>
<td>n/a</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>TailorBPEL</td>
<td>Yes</td>
<td>Yes</td>
<td>?</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>MASC</td>
<td>No</td>
<td>Limited</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Apache ODE Modification</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Fig. 7. Comparison of State of the Art

4.1.2 TRAP/BPEL

TRAP/BPEL utilizes service proxies to provide adaptability. Service proxies allow for a high degree of flexibility in the web services a calling process uses, and the TRAP/BPEL framework specifies well defined behaviour for a wide scale of adaptive actions.
A short-coming of this approach is that all BPEL processes that utilize the TRAP/BPEL framework need to be specially prepared at runtime, and outside of specially allocated points in the process, calling processes must remain static. Also, if the process uses stateful web services creating appropriate adaptive actions becomes highly complex [7]. A full technical publication of the TRAP/BPEL framework for developers is also not available, making a third-party implementation more difficult.

4.1.3 FlexBPEL

An application of the adaptive actions the FlexBPELs meta-object protocols make would theoretically allow for a very high degree of adaptability with regard to running BPEL processes. While the researchers working on FlexBPEL conclude that their meta-object framework could theoretically be integrated into most BPEL Engines [6], in their current design the process altering actions occurring within the reflection protocol would require non-standard extensions to BPEL engines. Also, the FlexBPEL framework does not yet provide behavioural consistency for fault and event handlers – highly common BPEL elements.
4.1.4 Dino Project

The most distinguishing feature of the Dino project is that it does not (inherently) use BPEL and WSDL for service composition, favouring instead the use of service brokers and OWL-S respectively. The benefits that this approach allows for in service modes and intelligent service selection could be potentially outweighed by the departure from what are acknowledged as industry standards [9]. This departure is detriment to developer familiarity and usability.

4.1.4 Manageable and Adaptable Service Compositions

MASC utilizes a somewhat similar approach to that of the TRAP/BPEL framework, emphasizing adaptive actions on a service level, and utilizing service proxies to switch between web services. The engine compatibility MASC offers is advantageous, providing wide-scale reusability of core processes across BPEL engine implementations.

MASC, like TRAP/BPEL suffers in flexibility when a modification to a core calling process is required, as all changes to the activities of a core BPEL process would require downtime. Also, MASC still lacks a full protocol specification, making an (equivalent) implementation difficult.
4.1.5 Apache ODE Modification

Mattanja Kern’s Apache ODE extensions are noteworthy in that it is the only published implementation of a BPEL engine that allows direct change to running processes.

Current limitations to the ODE extension are the removal and addition of partner web services from processes. An implementation of a BPEL process utilizing the modified Apache ODE engine, and supplemented by dynamic binding techniques has been attempted as a project goal, but was ultimately unsuccessful due to technical difficulties, as follows.

- There is a considerable lack of documentation to files that have been modified and added to standard Apache ODE engine
- Apache ODE is a very large project, over 600,000 lines of code, and over 10,000 individual files
  - Thus, full comprehension of the Apache ODE project is highly time-consuming
- There are contradictions in the building software required within the source code

Apache ODE is built with the apache designed, ruby based building software buildr. Buildr is subject to constant revisions, and does not maintain backwards-compatibility. The modified Apache ODE engine includes a buildfile that specifies buildr version 1.3, yet the
documentation for the relevant Apache ODE revision specifies buildr version 1.3.1. Neither 1.3, nor 1.3.1 or indeed any 1.3.x or 1.2.x version of buildr can fully build the engine. The most successful build of the modified Apache ODE engine has been with 1.4.x versions of buildr, but still meet with failure before completion even after considerable manipulation of the buildfile and source code.

It can be so concluded that without further documentation to utilize Apache ODE Modification a complete porting to the latest revision of the standard Apache ODE revision is the most sensible course of action for future projects. However, with 100,000+ extra lines of code, and considerable structural changes in the Apache ODE engine, this is a non-trivial task that could not be met within the scope of this report.

4.2 Dynamic Linking Implementations

To demonstrate two particular approaches that utilize dynamic linking two executable templates have been designed.

4.2.1 Dynamic Configuration

First we define two entities - parent processes, and child processes. There should be a single parent process and one or more child processes. Parent processes will remain static once they have been defined at design time. The developer provisions for change at design
time by identifying key points in the parent process where new behaviour can be added at runtime. Within these key points the developer defines a dynamic link, and activities to query a service repository and assign the resulting service to the dynamic link, as shown in figure 8.
The service repository is a stateful web service designed using Java Session Beans. The service repository includes a hash table, using string identifiers as keys to service endpoints. It provides methods to add, get, and delete services. With a parent process running a user can individually invoke a service repository adding and removing child processes to the repository as required.

Parent processes take a message and a service identifier as input. Upon reaching a query repository activity, the relevant service is resolved by the repository and the correct service is allocated to the dynamic link in the parent process.

Provided that a developer includes and does not corrupt the invocation and assign activities required by the dynamic configuration template, a parent process may be as large as the developer requires, allowing the full design breadth BPEL provides. Child processes are restricted in that they must have compatible WSDL interfaces (see 3.3). However, if suitably generic message types and operation names are defined, the restrictions of dynamic linking are not problematic. If processes required for an application have an interface that is

---

1 Recall that a BPEL process has an interface defined by WSDL. The Service Repository may then contain references to either “simple” web services, or BPEL processes, or a combination of both. Similarly the dynamic link of a parent process can refer to processes as well as web services.
incompatible we can define a linking process as an intermediary for the required process as shown in figure 9.

Fig. 9. A method to overcome non-compatible interfaces

4.2.3 Dynamic Adaptation with Fault Handling

The dynamic adaptation template uses a similar design - we define a parent process and child processes that the parent process calls. A developer utilising the template identifies critical web services/processes within the parent process that are subject to error. The template defines that we wrap calls to such critical services in a fault catching entity.

Upon any irregular behaviour by the critical service, the fault handler refers to a supplementary service. This supplementary service should contain equivalent behaviour to the standard service used\(^1\).

\(^1\) A additional message element informing the user that the standard service has failed is recommended but not required.
Chapter 5

Conclusions

The body of this work has been to find the most mature, practical approach to introducing adaptive actions to web services assemblies. A full solution must satisfy the following:

- Allows unrestrained change to the assembly
- Doesn’t require downtime for adaptation
- Is robust - can provide adaptive behaviour when required
- Has a well-defined usage and behavioural specification
- Is compatible with standard software

Fig. 9. The architecture of the dynamic adaptation template
From evaluating current solutions it is apparent that no published method satisfies all these criteria. The most practical approaches emphasize the introduction of adaptive behavior at a web service level (dynamic configuration, dynamic adaptation, MASC). Unfortunately service-oriented approaches still lack the ability to alter the logic of calling processes during runtime.

Solutions that propose the most flexible adaptability require changes to BPEL and its execution engine. This is a costly approach with a severe shortcoming in the necessary BPEL engine implementations. Also, the highly distributed manner of service-oriented computing is compromised with the introduction of incompliant elements.

To satisfy all requirements for adaptability, particularly compliancy, it is highly likely a new language specification is required (Swanson, 2009).

5.1 Future Work

The most obvious candidate for further investigation is Mattanja Kerns Apache ODE Modification, particularly porting his work to a current build of the Apache ODE engine. Using the results of this investigation as background, a new extension to BPEL could be specified and a compliant engine implemented allowing a full range of flexibility for adaptive web service systems.
An interesting further development could be made using the dynamic configuration and adaption templates by creating a “injection” application whereby users could supply a base BPEL file and with some basic configuration insert the necessary code to make it adaption ready. This would save interested developers time in utilizing the framework.

5.2 Knowledge Gained

During the course of this project the following skillsets were obtained:

- Competency in the use of web services, including
  - BPEL – Used to specify service assemblies
  - WSDL specification – required for web service interfaces
  - Java Business Integration – used as web service implementations throughout the project
  - XML, and XML schema – required for many stages of web service composition including messages, types, and language structures
- Ruby, necessary for understanding Apache building software

Furthermore, valuable experience was gained with regards to project management, public speaking, and general presentation skills.
References


