Weaving Aspects into Web Service Orchestrations

Carine Courbis
University College London
Department of Computer Science
Adastral Park
Martlesham Heath
IP5 3RE, UK
c.courbis@cs.ucl.ac.uk

Anthony Finkelstein
University College London
Department of Computer Science
Gower Street
London
WC1W 6BT, UK
a.finkelstein@cs.ucl.ac.uk

Abstract

Web Service orchestration engines need to be more open to enable the addition of new behaviours into service-based applications. In this paper, we illustrate how, in a BPEL engine with aspect-weaving capabilities, a process-driven application based on the Google Web Service can be dynamically adapted with new behaviours and hot-fixed to meet unforeseen post-deployment requirements. Business processes (the application skeletons) can be enriched with additional features such as debugging, execution monitoring, or an application-specific GUI.

Dynamic aspects are also used on the processes themselves to tackle the problem of hot-fixes to long running processes. In this manner, composing a Web Service 'on-the-fly' means weaving its choreography interface into the business process.

1. Introduction

With the advent of Web Service technologies [21]: WSDL (*Web Service Description Language*) to describe the business interfaces of the services; SOAP (*Simple Object Access Protocol*) to exchange messages between them, independently of the underlying protocol; and, UDDI (*Universal Description, Discovery, and Integration*) to publish and discover them; applications can be quickly composed across the network through firewalls by integrating loosely coupled services. There is a fundamental shift within businesses to a Service-Oriented Architecture (SOA). As the result of standardization, services from different systems and companies can be connected. Legacy and back-end systems can also be readily encapsulated into services and then integrated into applications. With Web Services, there is a layer of abstraction above the core business logic of the components.

The way Web Services interact with each other at the message level, including the execution order and the data flow, is called orchestration [16]. BPEL (*Business Process Execution Language*) [2], WSCI (*Web Service Choreography Interface*), and BPML (*Business Process Management Language*) are examples of Web Service orchestration languages or specifications. To offer business agility, Web Service orchestration engines need to be made more open. Two main requirements are to enable local code execution - code snippet as proposed in BPELJ [3] - and dynamic modifications of the processes to accomodate new customer needs and market conditions. Within this paper, we only deal with these types of adaptations. At the moment, features such as data conversion, execution monitoring, debugging, or an application-specific GUI can only be integrated into a process-driven application, if they are implemented as Web Services, making the process control flow more complex. Also, adapting a process can only be performed by stopping it, which is unacceptable especially for long-running processes. Steering a process is impossible.

To open up Web Service orchestration engines, we propose to embed aspect-weaving capabilities into their architectures. Aspect-Oriented Programming (AOP) [9] can be one solution to enhance and adapt business process execution. To demonstrate this, we have developed a BPEL engine prototype using this paradigm. Using this engine prototype, the behaviour and the structure of an application can be altered at runtime. Hot-fixes of business processes are possible as well as Web Service hot-deployment, specially useful for long running processes. In this way, new Web Services can be composed into the flow 'on-the-fly'. We have presented this approach, from a language engineering perspective, in [6]. However, in this paper, we focus on the Web Service perspective and gives a concrete example used through the paper to illustrate our approach. This example is a simple application based on the Google search engine Web Service[1] [5].

This paper is organised as follows. Section 2 introduces the business process of our example. Section 3 explains the main concepts of AOP, and presents our solution based on aspects to obtain adaptable executions and extensible business processes. The architectural details of our BPEL engine prototype are provided in Section 4. Then Section 5 presents the related work and we conclude the paper in Section 6.

2. An application based on the Google Web Service

This section describes the application used through the paper. First a quick overview of BPEL is provided.

2.1. BPEL

The de facto standard BPEL, currently submitted to the OASIS consortium, is the most well established orchestration technology for Web Services. Tools from BEA, IBM, Microsoft and Oracle to mention only the major companies involved in this market support BPEL. This XML-based workflow language is rather small [11] but is sufficient to handle typed variables with scopes, loops, conditional branches, synchronous and asynchronous communications, concurrent activities with correlated messages, transactions with compensations, and exceptions.

A BPEL business process is made of three main entities:

- The *partners* that abstractly represent the services involved in the composition.
- The *variables* used to manipulate the data (SOAP messages) exchanged between partners and to hold states of the business logic. XPath expressions can be used to access a part of a variable or to test conditions.
- The *activities* that describe the business logic. They can be basic such as invoking a Web Service or assigning a value to a variable, or structured such as executing a set of activities in sequence or in parallel.

[1]http://www.google.com/apis/

The process can then be interpreted by any compliant-BPEL engine. The process is itself a Web Service that can be used inside another business process (recursive composition). After Web Service deployment, the process is static as only the endpoints can be updated in the flow. Addressing business process dynamics and non-functional properties is out of the scope of BPEL.

2.2. The business process

To illustrate the benefits gained from using our open BPEL engine prototype, we have chosen to specify a simple process-driven application, based on well-known Google operations such as querying, spell-checking, and fetching cached Web pages. Google offers an alternative mechanism to access its operations via a Web Service for non-commercial use which restricts access to 1000 invocations a day per user. A free license key is assigned to each user.

To keep it simple and easy to understand, the business logic of our application only consists of obtaining information about the most relevant Web pages to a given query and, for each, displaying the cached Web page. The corresponding BPEL business process is given in Figure 1. It starts by declaring the partner and variables used as well as initialising the query (message) to send to the Google Web Service (first `assign` instruction). After the query, the iteration variable is initialised, as well as the number of items of the result and the Google key part of the message to fetch the cached web page by populating it with the key of the request message. Then, for each item of the result of the query, the Google cached web page URL is retrieved to fetch its HTML document and the iteration variable is updated.

To make this process useful, a GUI is needed. Otherwise the query values cannot be changed except by modifying the process and the cached Web page contents are retrieved for nothing as they are not stored or displayed. The only solution to plug the GUI into the process is to encapsulate it as a Web Service (a partner) and to add the corresponding `receive` and `reply` instructions. The `receive` instruction replacing the first `assign` will provide appropriate values for the query and the `reply` appended after the last `invoke` will send the cached Web page HTML document to display it. Any additional feature such as converting a message into another format, debugging, execution monitoring that are non-functional or low level orchestration matter can only be integrated into the process via a specific Web Service. Some BPEL engines such as the late Collaxa engine bought by Oracle offer the possibility to execute some code but this `exec` instruction

```
<?xml version="1.0"?>
<process name="GoogleSearch"
 targetNamespace="http://www.cs.ucl.ac.uk/bpel/"
 xmlns="http://schemas.xmlsoap.org/.../business-process/"
 xmlns:goo="urn:GoogleSearch"
 xmlns:bpws="http://schemas.xmlsoap.org"
 xmlns:xsi="http://www.w3c.org/2001/XMLSchema-instance">
<partnerLinks>
 <partnerLink name="GooglePartner"
  partnerLinkType="GooglePType" partnerRole="provider"/>
</partnerLinks>
<variables>
 <variable name="search"
  messageType="goo:doGoogleSearch"/>
 <variable name="searchResponse"
  messageType="goo:doGoogleSearchResponse"/>
 <variable name="cache"
  messageType="goo:doGetCachedPage"/>
 <variable name="cacheResponse"
  messageType="goo:doGetCachedPageResponse"/>
 <variable name="i" type="xsi:int"/>
 <variable name="query" type="xsi:string"/>
 <variable name="itemNb" type="xsi:int"/>
</variables>
<sequence>
 <assign>
  <copy>
   <from>
    <search>
     <key>0123456789</key>
     <q>ucl computer science</q>
     <start>1</start>
     <maxResults>5</maxResults>
     <filter>true</filter>
     <restrict/>
     <safeSearch>true</safeSearch>
     <lr/>
     <ie>latin1</ie>
     <oe>latin1</oe>
    </search>
   </from>
   <to variable="search"/>
  </copy>
 </assign>
 <invoke partnerLink="goo:GoogleSearchService"
  portType="goo:GoogleSearchPort"
  operation="goo:doGoogleSearch"
  inputVariable="search"
  outputVariable="searchResponse"/>
 <assign>
  <copy>
   <from expression="1"/>
   <to variable="i"/>
  </copy>
  <copy>
   <from variable="searchResponse" part="return"
    query="count(/return/resultElements/item)"/>
   <to variable="itemNb"/>
  </copy>
  <copy>
   <from variable="search" part="key"/>
   <to variable="cache" part="key"/>
  </copy>
 </assign>
 <while condition="bpws:getVariableData('i') &lt;=
                   bpws:getVariableData('itemNb')">
  <sequence>
   <assign>
    <copy>
     <from expression="concat(
      '/return/resultElements/item[',
      bpws:getVariableData('i'), ']/URL')"/>
     <to variable="query"/>
    </copy>
    <copy>
     <from variable="searchResponse" part="return"
      query="bpws:getVariableData('query')"/>
     <to variable="cache" part="url"/>
    </copy>
    <copy>
     <from expression="bpws:getVariableData('i')+1"/>
     <to variable="i"/>
    </copy>
   </assign>
   <invoke partnerLink="goo:GoogleSearchService"
    portType="goo:GoogleSearchPort"
    operation="goo:doGetCachedPage"
    inputVariable="cache"
    outputVariable="cacheResponse"/>
  </sequence>
 </while>
</sequence>
</process>
```

Figure 1. The BPEL business process based on the Google Web Service

is not compliant to BPEL.

Also, if its process were more complex, a spell-checking of the query words might be required, depending on earlier results, to avoid getting a void result. But hot-fixing (modifying the structure of) the process, without stopping it, is impossible in the orchestration engines available.

3. Open orchestration based on Aspect-Oriented Programming

We propose the use of aspects to leverage these problems and open the orchestration engines. First, this section explains the main concepts of AOP and then presents our approach to adapt process behaviours and extend their structures.

3.1. Overview of Aspect-Oriented Programming

AOP [9] is a recent programming paradigm that has emerged to complement object-oriented programming as objects have proven to be inadequate to capture crosscutting concerns such as logging or debugging. Concerns were usually scattered and application code was tangled. With aspects, they can be cleanly modularized, making development, maintenance, and reuse easier. An aspect is made of three parts: structure refinements (for example, adding a new attribute to a class), code fragments, and location descriptions to identify where to plug the code fragments in (the what, the when, and the where). In AspectJ [8, 19], the most mature AOP implementation for the Java programming language, these different parts are respectively called *intertype declarations*, *advice*, and *pointcuts*. The well-defined points in the program flow that can be selected by the pointcuts are called the *join points*. For example, a pointcut can identify the call to a method of a given class and an advice trigged by this pointcut can specify extra code to execute after this call.

The ultimate aim of AOP is to replicate the same execution as when the code of the business logic and the aspects were tangled. This composition is performed by an aspect weaver (see Figure 2), either by code transformation or by use of hooks, at compilation, deployment or execution time. Aspects that are woven at execution time are called dynamic aspects.

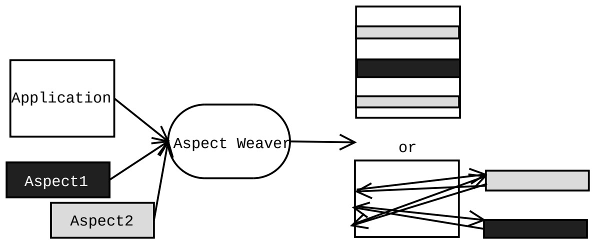


Figure 2. An aspect weaver

3.2. Adaptable behaviour

To offer business process adaptability, we propose the use of aspects on the top of a BPEL specification-compliant engine. As some of the adaptations cannot be foreseen at deployment time or need to be easily disabled, at any time, because they are performance-inefficient, there is a requirement for dynamic aspects.

With this technique, new behaviours (features or concerns) can be added to business processes at execution time. Each feature is a module that can be reused if non-process specific, and can be plugged in or unplugged dynamically without 'invasively' modifing the business processes. In the opposite, when the features are implemented as Web Services, business processes need to be modified to add the invocation to these Web Services. With BPELJ, each feature is scattered throughout the process, mixing Java and BPEL instructions, and cannot be added or removed at execution time. Debugging, execution monitoring and planning, Web Service selections after deployment time, and local code execution (for integration purposes between two services invocations) are examples of features that can be plugged into a business process without modifying the engine implementation as well as the structure of process itself.

We have chosen to develop our own aspect language dedicated to BPEL (a domain-specific aspect language) as with AspectJ, the pointcuts would be too low level, dependant on the engine implementation. With this abstraction from the engine, the aspects are resilient to the engine updates. It is also easier, for non-experts (business process designers), to develop aspects with this domain-specific aspect language. Also our implementation enables to have aspects that can be plugged in/out at runtime.

Such an aspect is made of two parts: the specifications of the pointcuts and a Java class. This class contains the methods (advice) used to adapt the process. It enables access to the current instruction being interpreted and to the BPEL interpreter environment as well as to any other aspect plugged in. The pointcuts to identify where the advice should be woven are written in XPath, a language specialized for addressing parts of XML documents. With this language, it is possible to weave advice before or after any BPEL instruction of a business process such as a fault handler, a `onMessage` event, or a scope. The identification of where to weave can be very precise if the aspect is process-dependant, or quite loose if the aspect can be applied on any business process.

With this solution, it was easy to embed seamlessly an application-specific GUI to the business logic of our example (see Figure 1). Figure 3 and Figure 4 are respectively the pointcut part and the advice part of the GUI aspect. This GUI (see Figure 5), according to the pointcuts, is initialized with the query values after the first `assign` of the process. The values of the query can be modified before the launch of the query by the user. Then, after each invocation of the `doGetCachedPage` operation, the GUI is updated with the retrieved Google cached Web page (on the screenshot, the cached UCL computer science home page which is the most relevant page to the `UCL Computer Science` query).

```
<?xml version="1.0"?>
<aspect name="uk.ac.ucl.cs.test.SearchGUI">
 <after where="//:assign[@name='initialisation']"
        methodName="updateRequest"/>
 <after
  where="//:invoke[@operation='goo:doGetCachedPage']"
        methodName="displayPage"/>
</aspect>
```

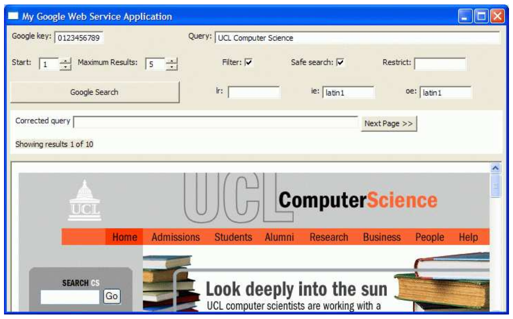
Figure 3. Pointcuts of the GUI, developed as an aspect

3.3. Extensible structure

To offer process hot-fixing capabilities, we also used dynamic aspects. In this case, the advice is written in

```
package uk.ac.ucl.cs.test;
...
public class SearchGUI extends AbstractEngineAspect {
...
 public void updateRequest() {
    // Display the Google search query contained in
    // the BPEL business process and enable its
    // modification before the WS invocation.
 }
 public void displayPage() {
  BPELEnv env = context.getBPELEnvironment();
  Variable cache = env.getVariable(new
      QName("http://www.cs.ucl.ac.uk/bpel/",
            "cacheResponse"));
  // Display the retrieved HTML document of the
  // cached web page and wait until the user asks
  // for the next one.
 }
}
```

Figure 4. Advice part of the GUI aspect


My Google Web Service Application

Google key: 0123456789
Query: UCL Computer Science
Start: 1
Maximum Results: 5
Filter: ✓
Safe search: ✓
Restrict:
lr:
ie: latin1
oe: latin1
Corrected query
Showing results 1 of 10

Figure 5. The GUI of our process-driven application

BPEL (the use of AspectJ was not possible). An important example of such hot-fixes is the composition, on demand of a new Web Service and thus the addition of its choreography interface into the process. The addition of this new partner with its activities, variables, endpoints, and wsdl file location into the business process corresponds to a concern.

In this manner, the structure of a process can be modified at runtime. A process can be steered, in other words, the end of the process can be changed depending upon results identified in earlier stages. This functionality is useful for long-running processes, for example a large Grid-based computational chemistry application [12].

With this mechanism, BPEL instructions can be inserted, deleted, or replaced as well as the partners, variables, exceptions, compensation or fault handlers. For example, we can insert into the the business process of our Google application the invocation of the spell-checker operation if the query has no result (see Figure 6).

```
Workflow aspect correctSpellingRequest
Members {
add <variable name="spellingCheck"
     messageType="goo:doSpellingSuggestion"/>
add <variable name="spellingResponse"
     messageType="goo:doSpellingSuggestionResponse"/>
}
Pointcuts {
   before //:while insert correctSpelling
}
Advices {
 correctSpelling
 <switch>
  <case condition="bpws:getVariableData('itemNb')==0">
   <assign>
    <copy>
     <from variable="search" part="key"/>
     <to variable="spellingCheck" part="key"/>
    </copy>
    <copy>
     <from variable="search" part="q"/>
     <to variable="spellingCheck" part="phrase"/>
   </assign>
   <invoke partnerLink="goo:GoogleSearchService"
    portType="goo:GoogleSearchService"
    operation="goo:doSpellingSuggestion"
    inputVariable="spellingCheck"
    outputVariable="spellingResponse"/>
   <assign>
    <copy>
     <from variable="spellingResponse" part="return"/>
     <to variable="search" part="q"/>
    </copy>
   <assign>
   <invoke partnerLink="goo:GoogleSearchService"
    portType="goo:GoogleSearchPort"
    operation="goo:doGoogleSearch"
    inputVariable="search"
    outputVariable="searchResponse"/>
   <assign>
    <copy>
     <from variable="searchResponse" part="return"
      query="count(/return/resultElements/item)"/>
     <to variable="itemNb"/>
    </copy>
   </assign>
  </case>
 </switch>
}
```

Figure 6. An example of a business process aspect: the insertion of the spell-cheking operation

4. Architecture

Our BPEL engine prototype is an interpreter, implemented using the visitor design pattern [7, 13]. It contains one `visit` method for each BPEL instruction and traverses, from top to bottom, the ASTs (*Abstract Syntax Trees*) that represent the BPEL documents. These trees are not only strictly typed to meet the pattern requirements but are also based on the DOM API to enable XPath selections of nodes, which is useful for the implementation of our two BPEL aspect languages.

The two BPEL aspect weavers are themselves aspects that can be plugged in or unplugged from the BPEL engine. The BPEL engine code is totally independent from them and is compliant with the BPEL specifications. More details about the visitor design pattern implementation we are using and its aspects can be found in [14].

When an aspect to adapt the behaviour of a business process is plugged in, it is registered and the different nodes of the process identified by the XPath expressions (pointcuts) are annotated with the aspect name and the name of the method to execute. Before and after interpreting an instruction, our system checks if there is any annotation and calls the method to execute (advice) if this aspect is still registered. Unplugging an aspect only means removing the aspect from the registry.

When an aspect to modify the structure of a business process is plugged in, all activities in the engine are suspended to perfom the transformations on the process. Also, members such as partners or variables may be added or removed to/from the environment of the interpretation. Additionally, the annotations of the other aspects already plugged in should be propagated onto any new BPEL instruction added by insertion or replacement.

5. Related work

The idea of applying dynamic aspects into component architectures to have more flexible and adaptable applications interests many research projects [1, 15, 18]. For example, the JAsCo infrastructure [18] enables the execution of component-based applications, which can be dynamically adapted by aspects. The components (Java Beans) when loaded into the infrastructure are modified to insert traps at every public method which invoke the aspect weaver when they are called.

Based on the JAsCo aspects, a Web Service Management Layer (WSML) [22] was developed to monitor and adapt Web Service applications. Dynamic selection of Web Service is available as well as hot-swapping if any service is down or if the selection policy has changed. We believe we will be able to develop a similar layer transparently on the top of our engine with aspects. If we manipulate workflows with transactions, however, hot-swapping of Web Services may not be possible at all time. Recent work [4], AO4BPEL, also proposes to use dynamic aspects in the BPEL business process context but only for Web Service composition purposes. Hot-fixes in the general sense of the term (composition being only a particular case) and execution adaptations such as monitoring are not taken into consideration.

Using aspects on SOA will make it possible, for example, to check constraints (design by contracts), such as the ones proposed in the Web Service Offerings Language (WSOL) [20]. This language might be used in our BPEL engine to assign contraints and then be implemented as aspects.

To address the problem of automatic selection and composition of Web Services to fulfill a task, OWL-S (formerly DAML-S) [10] proposes the use of semantic descriptions. These descriptions will then be manipulated by different agents or software.

The main problem with transforming the workflow at runtime is to ensure its stability. Despite the changes, code consistency and structural correctness should be maintained. Formal models to perform dynamic structural changes may be used such as ADEPT [17].

6. Conclusion

Enabling additions of new features and modications of business process at runtime are two requirements to make Web Service orchestrations capable of evolving to changes.

We illustrate how, in a BPEL engine with aspect-weaving capabilities, a process-driven application based on the Google Web Service can be dynamically adapted with new behaviours and hot-fixed to meet unforeseen post-deployment requirements. With this solution, composing a new Web Service in the workflow is also possible. The benefits of these adaptation mechanisms outweigh, we believe, the potential performance impact. Furthermore, this potential performance impact associated with the weaving mechanisms is not comparable with that due to remote service invocations over the Internet.

We plan to continue the implementation of our engine prototype to handle most of the BPEL instructions only for our research experimentations. But we do not plan to deal to the complexities and subtleties of certain measures of BPEL such as dead path elimination. We also wish to investigate how the stability of the system can be ensured when the process is transformed.

7. Acknowledgements

The work is supported by the Generative Software Development project funded by BT Group. Anthony Finkelstein is grateful for support from an IBM faculty partnership award. The authors want to thank PATRIK MIHAILESCU from BT for his comments on this article.

References

[1] ObAsCo (Objects, Aspects, and Components) Research Group. `http://www.emn.fr/x-info/obasco/`.

[2] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, and I. Trickovic. Business Process Execution Language for Web Services version 1.1. Technical report, BEA, IBM, Microsoft, SAP, Siebel Systems, May 2003. http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/.

[3] M. Blow, Y. Goland, M. Kloppmann, F. Leymann, G. Pfau, D. Roller, and M. Rowley. *BPELJ: BPEL for Java*. BEA and IBM, March 2004. white paper, http://www-106.ibm.com/developerworks/java/library/j-diag0925/.

[4] A. Charfi and M. Mezini. Aspect-Oriented Web Service Composition with AO4BPEL. In L. J. Zhang, editor, *In Proceeding of the European Conference on Web Services (ECOWS'2004)*, volume 3250 of *LNCS*, Erfurt, Germany, September 2004. http://www.st.informatik.tu-darmstadt.de/database/publications/data/cha%rfi-mezini-ecows-04.pdf?id=94.

[5] N. Chase. Building Web service applications with the Google API. IBM developerWorks, tutorial, May 2002. http://www-106.ibm.com/developerworks/edu/ws-dw-wsgoog-i.html?S_TACT=10%4AHW04&S_CMP=EDU.

[6] C. Courbis and A. Finkelstein. Towards Aspect Weaving Application. In *The 27th International Conference on Software Engineering (ICSE'2005)*, Saint-Louis, Missouri, USA, May 2005. ACM press. http://www.cs.ucl.ac.uk/staff/C.Courbis/papers/icse2005.pdf.

[7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns*. Addison-Wesley Pub Co, January 1995. ISBN 0201633612.

[8] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold. An Overview of AspectJ. In J. L. Knuden, editor, *Proceedings of European Conference on Object-Oriented Programming*, volume 2072 of *LNCS*, pages 327–355, Budapest, Hungary, June 2001. http://www.cs.ubc.ca/~gregor/kiczales-ECOOP2001-AspectJ.pdf.

[9] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Videira, and J.-M. Loingtier. Aspect-Oriented Programming. In M. Aksit and S. Matsuoka, editors, *Proceedings of the 11th European Conference on Object-Oriented Programming*, volume 1241 of *LNCS*, pages 220–242, Jyväskylä, Finland, June 1997. Springer-Verlag. http://www.cs.ubc.ca/~gregor/kiczales-ECOOP1997-AOP.pdf.

[10] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara. OWL-S 1.1: Semantic Markup for Web Services. Technical report, OWL-S Coalition, November 2004. http://www.daml.org/services/owl-s/1.1/overview/.

[11] N. Mukhi. Reference guide for creating BPEL4WS documents. Technical report, IBM, November 2002. http://www-106.ibm.com/developerworks/webservices/library/ws-bpws4jed/.

[12] H. Nowell, B. Butchart, D. S. Coombes, S. L. Price, W. Emmerich, and C. R. A. Catlow. Increasing the Scope for Polymorph Prediction using e-Science. In *the 2004 UK E-Science All Hands Meeting (AHM)*, pages 967–971, Nottingham, UK, September 2004. UK Engineering and Physical Science Research Council. http://www.cs.ucl.ac.uk/staff/w.emmerich/publications/AHM04/final_nowel%l.pdf.

[13] J. Palsberg and C. B. Jay. The Essence of the Visitor Pattern. In *Proceedings of COMPSAC'98, 22nd Annual International Computer Software and Applications Conference*, pages 9–15, Vienna, Austria, August 1998. http://www.cs.ucla.edu/~palsberg/paper/compsac98.pdf.

[14] D. Parigot, C. Courbis, P. Degenne, A. Fau, C. Pasquier, J. Fillon, C. Held, and I. Attali. Aspect and XML-oriented Semantic Framework Generator: SmartTools. In M. van den Brand and R. Lämmel, editors, *ETAPS'2002, LDTA workshop*, volume 65 of *Electronic Notes in Theoretical Computer Science (ENTCS)*, Grenoble, France, April 2002. Elsevier Science. http://www.elsevier.nl/gej-ng/31/29/23/117/52/33/65.3.009.pdf.

[15] R. Pawlak, L. Seinturier, L. Duchien, and G. Florin. JAC: A Flexible Solution for Aspect-Oriented Programming in Java. In A. Yonezawa and S. Matsuoka, editors, *Metalevel Architectures and Separation of Crosscutting Concerns: Third International Conference, Reflection'01*, volume 2192 of *LNCS*, pages 1–24, Kyoto, Japan, September 2001. http://jac.aopsys.com/papers/reflection.ps.

[16] C. Peltz. Web Services Orchestration - a review of emerging technologies, tools, and standards. Technical report, HP, January 2003. Technical white paper, http://devresource.hp.com/drc/technical_white_papers/WSOrch/WSOrchestra%tion.pdf.

[17] M. Reichert and P. Dadam. ADEPTflex - Supporting Dynamic Changes of Workflow without Loosing Control. *Intelligent Information Systems special issue one Workflow Management Systems*, 10(2):93–129, March/April 1998. http://www.informatik.uni-ulm.de/dbis/01/staff/reichert/papers/journals%/reda98c.pdf.

[18] D. Suvée, W. Vanderperren, and V. Jonckers. JAsCo: an Aspect-Oriented approach tailored for Component Based Software Development. In *Proceedings of the 2nd international conference on Aspect-oriented software development*, pages 21–29, Boston, USA, March 2003. http://ssel.vub.ac.be/jasco/papers/aosd2003.pdf.

[19] The AspectJ Team. *The AspectJ Programming Guide*, AspectJ 1.2 edition. http://dev.eclipse.org/viewcvs/indextech.cgi/~checkout~/aspectj-home/do%c/progguide/index.html.

[20] V. Tosic, B. Pagurek, and K. Patel. WSOL - A Language for the Formal Specification of Various Constraints and Classes of Service for Web Service. In *The International Conference On Web Services, ICWS'03*, pages 375–381, Las Vegas, USA, June 2003. CSREA Press. http://www.sce.carleton.ca/netmanage/papers/TosicEtAlResRepNov2002.pdf.

[21] A. Tsalgatidou and T. Pilioura. An Overview of Standards and Related Technology in Web Services. *Dis-*

tributed and Parallel Databases, special issue on e-services, 12:135–162, 2002. Kluwer Academic Publishers, `http://www.di.uoa.gr/~afrodite/PADP2002.pdf`.

[22] B. Verheecke and M. A. Cibràn. AOP for Dynamic Configuration and Management of Web Services. In *In Proceeding of the International Conference on Web Service Europe (ICWS-Europe'03)*, volume 2853 of *LNCS*, Erfurt, Germany, September 2003. `http://ssel.vub.ac.be/wsml/papers/Verheecke_Cibran_ICWS03.pdf`.