Conflict Resolution in Web Service Federations

Veruska R. Aragão and Alvaro A.A. Fernandes

Department of Computer Science
University of Manchester
Manchester M13 9PL, UK
{v.aragao,a.fernandes}@cs.man.ac.uk

Abstract. Web services are expected to become the foundation of highly-dynamic distributed business computing architectures. By building upon the Internet-based technologies that changed the face of business computing over the last decade, web services are a significant step towards a truly universal infrastructure for e-business. Focusing on the case of opportunistic virtual enterprises, this paper first considers the widely-held view that they can be modelled by web service federations and then highlights the fact that, in that case, similar problems to those faced in attempts to federate databases will also stand in the way of virtual enterprise formation. A lightweight, unobtrusive approach is described to improve the chances that web service federations can be formed unimpededly, thereby reducing the cost of virtual enterprise formation and operations. Therefore, the paper contributes an enabling technology to facilitate an important, and highly dynamic, kind of e-business.

Keywords: Web services, Virtual Enterprises, Dynamic business process adaptation and composition, Mediation middleware, Rule-based conflict resolution of heterogeneities.

1 Introduction

The greatest promise of the web services approach [8] to business computing is that it will make it significantly easier for businesses to orchestrate different services (possibly from different, autonomous providers) into complete business processes that can be enacted over an Internet-based fabric.

The kind of enterprise that such business processes would give rise to have been called *virtual enterprises* (VEs) [13,17]. VEs can be understood as a federation of business processes. The federation is the outcome of an organizational process by which different individual, autonomous enterprises pool together their complementary competences. This complementarity gives rise to a new, composite enterprise whose pooled business competences allow it to pounce on business opportunities unattainable by its members in isolation.

This organizational process can be modelled fairly directly under the web services approach, as follows. Assuming competences to be organizationally captured in business processes, web services can be seen to capture and expose the

M. Jeckle and L.-J. Zhang (Eds.): ICWS-Europe 2003, LNCS 2853, pp. 109–122, 2003.
© Springer-Verlag Berlin Heidelberg 2003

latter concretely. In the context of VE formation, publishing a web service onto a public registry corresponds to submitting the competences it embodies to a pool of candidates. The federation process from which a VE arises is captured in two steps: firstly, all the web services that comprise the desired configuration of competences need to be found in public registries; secondly, such a configuration is given computational expression as a workflow in some web service orchestration language. Once the federation is formed, the operation cycle of the new organization corresponds to the enactment of the composite workflow characterizing the complete business process which the federation was formed to offer.

Computationally, the requirements for the web services approach include:

1. each provider should be in a position to deliver web services quickly and at an economical cost, in full awareness that the federation may be disassembled quickly;
2. each provider benefits the most if the web service is kept stable (i.e., requires no adjustment for participation) while still allowing that web service to participate in many other federations both at once and over time.

The web services technologies currently available [18] can be said to satisfy Requirement (1) above, but not Requirement (2). Web services build on widely-adopted Internet standards and are themselves standardized with a very light hand. In this way, many impediments to quick phase-in and phase-out are removed, and participation costs are kept comparatively low. However, in the scenarios covered in this paper, because the returns may be over short periods only, the ability of web services to participate *without change* in many federations both at once and over time is highly desirable. Unfortunately, this cannot be ensured by standardization alone. There are impediments for the federation of stable components whose removal requires not just standardization, but *mediation* [19,20] as well.

Given the view proposed here of VEs as federations of computational resources embodied in web services, it turns out that the well-studied impediments described in [9,10] to federating data resources [3] can be used to show both why standardization alone does not suffice to fulfill Requirement (2) above and what kind of solution is needed, as follows. Consider this: databases that adhere to the standard represented by the relational data model are not thereby protected from impediments to federation. As shown in [10], many forms of schematic and data heterogeneities conspire against participation without change of a particular database in many database federations. Since it is as undesirable to redesign and re-implement a database for each federation as it would be for a web service, solutions for schematic and data heterogeneities have always been eagerly sought. One effective and efficient approach to heterogeneity problems in database federations involves the definition of views over the data [12]. Such views act as mediators between the reality of an autonomous member database and the needs of each of the federated databases of which it is a member [9].

The remainder of the paper is structured as follows. Section 2 introduces an application scenario to illustrate issues, challenges and solutions. Section 3 describes a taxonomy of heterogeneities that are likely to arise in web service

federations. Then, in Section 4, the contributions of the paper are presented, viz., an approach to the resolution of the heterogeneity conflicts described in Section 3. Section 5 considers related work, and, finally, Section 6 draws some conclusions.

2 Application Scenario

Consider the business scenario shown as an extended data flow diagram in Figure 1. It shows three small, specialist providers (viz., PDP, ET and TS) whose web services (among others not shown) are being orchestrated into a customer-facing web service federation that computationally captures a VE called PET.

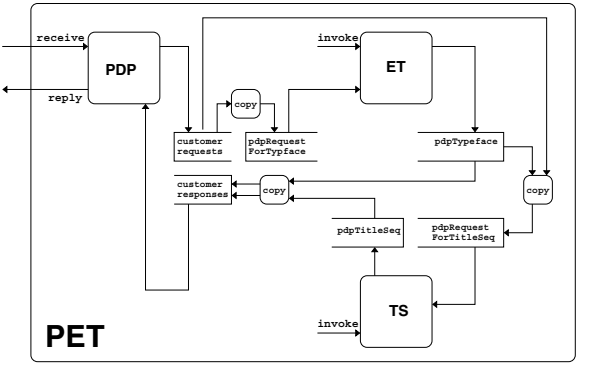


Fig. 1. The PET Virtual Enterprise

PDP specializes in pulling together the various skills that go in the the production design of television programs. ET specializes in the design of bespoke, one-off, copyrighted electronic type faces. TS specializes in the design of title sequences. The business opportunity behind PET was spotted by PDP, viz., a contract for the production design of a new television series in which the use of sophisticated, distinctive electronic typography plays a major role. PDP, therefore, proposed to ET and TS (and possibly others not shown) the formation of the PET VE for the duration of the production design process. The PET virtual enterprise starts when PDP, which leads the VE, receives a request from a customer to produce a new television program. Thus, PDP invokes ET to produce the typeface for that program. As soon as ET creates the typeface PDP invokes TS to produce the title sequence of the program using the typeface delivered. It is assumed that PDP invokes other services to complete the production of the new television program. However, the services described here are enough to point out the relevant issues in our discussion.

This paper assumes that business opportunities have short windows of opportunity and lead to short lifespans, therefore, in practice, companies such as

the above are likely to be involved in many such VEs at any one time, routinely changing exactly which VEs they are currently members of and for how long, although, for simplicity, this multiple participation is not shown in Figure 1[1] The WSDL [5] documents published by PDP and ET are given in Figures 2 and 3, respectively (TS's is omitted for reasons of space).

They reveal several barriers for integration, i.e., they contain conflicts that require resolution before the described services can be orchestrated into the desired VE. In general, such conflicts are unavoidable in practice, because the web service designers have many routes to implementation and must make choices. A conflict arises whenever such a commitment (e.g., to choice of names, or of types, etc.) by the implementer of a web service W differs from the corresponding commitments by the implementers of web services $W_1, \ldots, W_n$ that might bind to W. So, while at a higher level of abstraction (i.e., the design level) two highly cohesive services can be integrated, concretely they cannot couple. A resolution mechanism is required that reconciles the two concrete choices by mapping either one to the other, thereby smoothing out what is, in essence, irrelevant detail that only becomes significant due to its standing in the way of orchestration. For example, in Figures 2 and 3, a type face specification by PDP (i.e., `typefaceSpecs`) has inessential variations with ET's (i.e., `typeface`). Conflicts such as these may prevent the ET web service to be found in web service registries by PDP. Even if it is found, as is being assumed, some conflict resolution is still required before PDP can bind to it. The research challenge is to decide how, where and when to effect this conflict resolution.

Since this paper is concerned with web service federation as a computational model for opportunistic VEs, both publishing and matching of WSDL documents are assumed to have been carried out, as preliminary steps corresponding to VE formation. Thus, the challenge is, more specifically, that of resolving heterogeneity conflicts during the enactment of the workflows that express the VE. What is needed is an unobtrusive mechanism that prevents inessential implementation dissimilarities from defeating the essential design similarities that justified the attempt at orchestration. By unobtrusive is meant that, ideally, the conflict resolution mechanism should not be sucked into the orchestration itself, not should it require maintenance (in the form of re-design or re-implementation) of either party in the interaction that is being disrupted.

Web service federations lack an unobtrusive mechanism to express conflict resolution. The solution proposed here is inspired by the use of views for the same purpose in database federations. In Section 4, a restricted, well-behaved class of event-condition-action (ECA) rules [14] is defined that resolves a significant set of conflicts arising in web service federations.

[1] The web service federation in Figure 1 is expressed in the BPEL4WS web orchestration language, (for the BPEL4WS specification, see [`http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/`]) in an extended version of this paper

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<definitions targetNamespace="http://www.productionDesigns.com"
  xmlns="http://schemas.xmlsoap.org/wsdl"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  xmlns:tns="http://www.productionDesigns.com"
  <types>
    <xsd:schema>
      <xsd:element name="typeface"      type="xsd:string"/>
      <xsd:element name="titleSeqSpecs" type="xsd:titleSeqSpec"/>
      <xsd:element name="titleSeq"      type="xsd:string"/>
      <xsd:element name="prodSpecs">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="typefaceSpecs" type="typefaceSpecs"/>
            <xsd:element ref="titleSeqSpecs"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="prodDesign">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element ref="typeface"/>
            <xsd:element ref="titleSeq"/>
            ...
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:complexType name="typefaceSpecs">
        <xsd:sequence>
          <xsd:element name="varieties" type="typefaceStyle" />
          <xsd:element name="serif"    type="xsd:boolean" default="true"/>
          ...
        </xsd:sequence>
      </xsd:complexType>
      <xsd:complexType name="titleSeqSpecs">
        <xsd:sequence>
          <xsd:element ref="typeface" />
          <xsd:element ref="titleSeq" />
        </xsd:sequence>
      </xsd:complexType>

      <xsd:simpleType base="xsd:string" name="typefaceStyle">
        <xsd:enumeration value="bold"/>
        <xsd:enumeration value="italic"/>
        <xsd:enumeration value="slanted"/>
      </xsd:simpleType>
    </xsd:schema>
  </types>
  <message name="customerRequest">  <part name="prodSpecs"            element="prodSpecs"/>
  </message>
  <message name="customerResponse"> <part name="prodDesign"           element="prodDesign"/>
  </message>
  <message name="requestToET">      <part name="requestTypeface"      type="typefaceSpecs"/>
  </message>
  <message name="responseFromET">   <part name="typeface"             element="typeface"/>
  </message>
  <message name="requestToTS">      <part name="requestTitleSeq" type="titleSeqSpecs"/>"
  </message>
  <message name="responseFromTS">   <part name="titleSeq"        element="titleSeq"/>
  </message>
</definitions>
```

Fig. 2. WSDL Document for the PDP Web Service

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<definitions targetNamespace="http://www.electronicTypeface.com"
  xmlns="http://schemas.xmlsoap.org/wsdl"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  xmlns:tns="http://www.electronicTypeface.com"
  <types>
    <xsd:schema>
      <xsd:complexType name="typeface">
        <sequence>
          <xsd:element name="serif"     type="choices"/>
          <xsd:element name="styles"    type="typefaceStyle"/>
          <xsd:element name="lowerCase" type="xsd:boolean" default="false"/>
        </sequence>
      </xsd:complexType>
      <xsd:simpleType base="xsd:string" name="choices">
        <xsd:enumeration value="yes"/>
        <xsd:enumeration value="no"/>
      </xsd:simpleType>
      <xsd:simpleType base="xsd:string" name="typefaceStyle">
        <xsd:enumeration value="bd"/>
        <xsd:enumeration value="it"/>
        <xsd:enumeration value="slt"/>
      </xsd:simpleType>
    </xsd:schema>
  </types>

  <message name="propertiesTypeface"> <part name="designSpecs" type="typeface"/>
  </message>
  <message name="typeface">           <part name="characters" type="xsd:string"/>
  </message>

  <portType name="typefacePortType">
    <operation name="designTypeFace">
      <input message="tns:"propertiesTypeface"/>
      <output message="tns:typeFace"/>
    </operation>
  </portType>

  <binding name="etBinding" type="tns:typefacePortType">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="designTypeFace">
      <soap:operation soapAction=""/>
      <input>
        <soap:body
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding"
          namespace="urn:typeface" use="encoded"/>
      </input>
      <output>
        <soap:body
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding"
          namespace="urn:typeface" use="encoded"/>
      </output>
    </operation>
  </binding>

  <service name="etService">
    <port binding="tns:etBinding" name="etPort">
      <soap:address location="http://et.resources.net:80/soap"/>
    </port>
  </service>
</definitions>
```

Fig. 3. WSDL Document for the ET Web Service

3 Web Services Heterogeneities

The application scenario in Section 2 is now used to illustrate how web service interactions, and hence the formation of opportunistic VEs, may be impeded by heterogeneities. This section presents a coarse classification of heterogeneities that are probable in web service federations. The classification is inspired by the one introduced in [9,10] for database federations. It assumes that the messages exchanged are SOAP messages [5] invoking WSDL-defined operations on values whose types are defined using XML Schema [5]. In the context of this paper, the heterogeneities of interest are those that manifest themselves in SOAP messages (since the conflict resolution mechanism described in Section 4 acts on those), although there is no reason why the approach could not be deployed with greater scope. The heterogeneities that arise in this context can be categorized into *description-level heterogeneities* and *value-level heterogeneities*.

Description-Level Heterogeneities arise when two web services describe differently one or more of the elements in the message they exchange. They can be further split into *name*, *type*, and *cardinality* heterogeneities.

Name Heterogeneity occurs when two elements with the same semantics are assigned different concrete names or when the same concrete name is given to two elements with different semantics. In the former case, the implementors assign different labels to the same design concept, thereby impeding federation which would be unproblematic otherwise. In the latter case, the implementors assign the same label to different design concepts, thereby misrepresenting real federation impediments. As an example of the first case, PDP uses the name `varieties` in Figure 2 for what ET calls `styles` in Figure 3. Both names denote the same concept, and hence, the same set of values. As an example of the second, PDP might assign the name `price` to an attribute recording price alone while ET assigns it to an attribute recording price plus tax. Resolving such conflicts is necessary for the VE to function.

Conflict resolution in web service federations can be done in two contrasting ways. The first way is disruptive and involves maintenance intervention in the implementation of services. It is currently the only one available (for this and the other conflicts described in the remainder of this section). The second way is contributed by this paper: it is based on non-disruptive mediation and allows the implementation of services to remain as they are. In the first way, the participants agree which one among them is going to adopt the other's preferred description. One might think that all that would remain would be to change the WSDL document. However, this may open up previously resolved conflicts in other VEs that this VE member may be involved in: those other VEs rely on the WSDL as it stood before this conflict arose. Because of this, it is necessary to import, as additional internal logic, into the web service the reconciliation of the heterogeneity (roughly, a branch for each such conflict in a case analysis). Besides going against best practice in software design (which recommends keeping cohesion levels highs and coupling levels low), this approach is likely to be costly, time consuming and risky (insofar as every maintenance episode risks injecting faults into the implementation). The second way of resolving such

conflicts, as well as those in the remainder of this section, is made possible by the contributions reported in this paper and is described in detail in Section 4.

Type Heterogeneity occurs when two elements with the same semantics are assigned different concrete (built-in or used-defined) types or when the same concrete type is assigned to two elements with different semantics. As an example of the first case, in the scenario in Section 2, PDP and ET assign different concrete types to represent the *serif* abstract type. The former uses the `boolean` built-in XML Schema type while the latter goes for a user-defined type with domain {`yes, no`}.

Cardinality Heterogeneity occurs when different numbers of elements are used to describe what is, semantically, the same information. One special case is *arity heterogeneity*, another special case arises by the interplay of XML Schema constraints such as `minOccurs` and `maxOccurs`.

As an example of arity heterogeneity, in the scenario in Section 2, modulo name heterogeneity, ET requires three elements to specify a type face, viz, {`styles, serif, lowercase`}. However, PDP makes no provision for the specification of a `lowercase` element. Thus, from the viewpoint of ET there is one element missing in PDP-originated type face specifications.

Value-Level Heterogeneities arise when two web services define differently one or more of the values in the message they exchange even as they are homogeneous at the description level. They can be further split into *representational* and *interpretational* heterogeneities.

Representational Heterogeneity occurs when the same datum is concretely represented differently. As an example, in the scenario in Section 2, PDP represents the style *bold* with the string `bold`, while ET represents it with `bd`.

Interpretational Heterogeneity occurs when a different interpretation is given to the same concretely represented value. As an example, PDP may interpret `size` to be in pica, while ET may interpret it to be in points (where there are twelve points in a pica).

4 Resolution of Heterogeneity Conflicts

For the cases considered in this paper, the state of the art is that when conflicts arise in web service federations, their resolution requires maintenance intervention in the implementation of services. Especially In the case where one web service must participate in many federations, it is unlikely that the interface exposed by that web service will be exactly as expected by its potential users. The web services approach postulates that once the interface is obtained, the potential user is responsible for abiding by the service description embodied in the corresponding WSDL document. In other words, the requester has to factor into the code of the client application the generation of conformant messages to the server web service. However, in cases where opportunities are volatile, and hence the cost of quick client adjustment is high, unless there is an unobtrusive adjustment mechanism, business will be lost. If there were a mechanism that allowed, without code maintenance, either the server to adjust to many clients or

a client to adjust to many servers (or both), then web service federations would be more suited still for modelling dynamic, opportunistic VEs. This section describes exactly such an unobtrusive and non-intrusive mechanism to overcome the heterogeneities described in Section 3.

The mechanism is concretely implemented as a mediator component in the SOAP engine (e.g., Apache Axis [7]) that underpins web service invocation. It is referred to as a *personalization framework* insofar as it allows the adjustment of concrete interaction events so as to take into account specific characteristics of the interacting parties. In this way, it is analogous to the concept of a *view* in database settings, whereby a particular user can be provided with specific retrieval possibilities that, ultimately, may be exclusive to them. This analogy indicates that, as shown below, the use of views for conflict resolution in database federations can be emulated by personalization rules in web service federations.

The specification of personalization processes takes the form of a set of personalization rules. Such rules are processed by a personalization engine that is placed as the first component in the global handler chain (see Figure 4) for incoming messages (or the last for outgoing messages) within the SOAP engine of the party whose wish it is to personalize some (or all) of its interactions. In the specific context of this paper, personalization rules express conflict resolution strategies, and personalization engines act as mediators that allow the services exposed by an organization to remain unmodified even as they face many different heterogeneity conflicts across many different VE configurations.

Personalization Rules. A personalization rule is an event-condition-action (ECA) rule (see [14] for a comprehensive collection of research papers on the topic). An ECA rule has the general form: `on` ⟨event⟩ `if` ⟨condition⟩ `do` ⟨action⟩.

In a personalization rule, an *event* occurs when a SOAP message is received by the targeted SOAP engine in the host. Syntactically, an event is either the constant `true` or an XPath [5] expression that queries the header of the SOAP message (for features in the interaction such as its purpose, its originator, the date, the time, etc.). Semantically, an event causes a rule to *trigger* if it is `true` or else the XPath expression returns a non-empty result.

Once a rule is triggered, its *condition* is evaluated. Syntactically, a condition is either the constant `true` or an XPath expression (possibly compound, using the standard Boolean connectives) over the system environment that is visible through the host SOAP engine (including, e.g., the host organization's database systems). Semantically, a condition causes a rule to *fire* if it is `true` or else the XPath expression returns a non-empty result.

Once a rule is fired, its *action* is executed. Syntactically, an action is an XSLT [5] program. Semantically, an action changes the body of the incoming SOAP message and hands over the message, thus personalized, to the next handler in the chain.

Personalization Engines. The Axis SOAP Engine is designed as chain of *handler chains*. There are three standard handler chains for requests and three for responses: they are paired to deal, respectively, with matters of *transport*, matters

that are *global* to all web services, and matters that are specific to a *service*. At the sink of a request and at the source of a response lies the *target web service*. This is depicted in the top part of Figure 4. The role of each chain is to provide a sequence of opportunities for manipulation of incoming and outgoing messages. The personalization engine proposed here is implemented as an additional handler chain that is deployed as prefix chain of the standard global handler, on the way in, and as a suffix chain, on the way out. In the top part of Figure 4, the personalization engine is depicted as shaded bands.

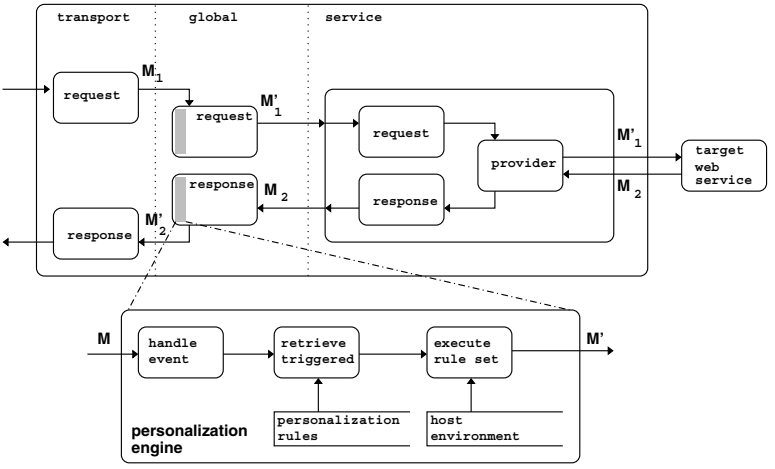


Fig. 4. A SOAP Engine with Personalization

The bottom part of Figure 4 shows the internal structure of the *personalization engine* (the shaded bands) as a handler chain deployed inside the global handlers. The **handle event** component receives the original SOAP message M, identifies the sender and appends this identification to the message context. The **retrieve triggered (rules)** identifies, within the rules associated with the sender, those whose event part match the message received (or are event-independent, i.e., have the event part equal to `true`).

The matching is effected against an underlying XPath-compliant store (e.g., Apache Xindice [6]).

The retrieved triggered rules are passed on to the `execute rule set` component. Note that because the event part simply denotes the arrival of a message from a sender, the evaluation of the rule set cannot generate new events. This, in turn, means that, at the evaluation stage, the ECA rules become *productions* (i.e., CA rules), losing their event part. Thus, a CA-rule evaluation engine (e.g., JESS [11]) suffices to evaluate them. It is the execution of the rule set that effects personalization of the original message M into its personalized version M'. The rule engine proceeds in the classical way, except that a condition is evaluated by matching the XPath expression that captures it against (a suitable representation of the relevant features of) the host environment and an action is performed

```
<personalizationRule> <!-- - - - - - - - - reconciling name heterogeneity -->
  <on> //interaction[sender="http://www.productionDesigns.com"]  </on>
  <if> //varieties  </if>
  <do>
    <xsl:stylesheet version="1.0"
                    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
      <xsl:output indent="yes" method="xml"/>
      ...
      <xsl:template match="varieties">
        <xsl:element name="styles">
          <xsl:value-of select="."/>
        </xsl:element>
      </xsl:template>
    </xsl:stylesheet>
  </do>
</personalizationRule>
<personalizationRule> <!-- - - - - - - - - reconciling cardinality heterogeneity -->
  <on>   //interaction[sender="http://www.productionDesigns.com"]  </on>
  <if>   not(//lowercase) </if>
  <do>
    <xsl:stylesheet version="1.0"
                    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
      <xsl:output indent="yes" method="xml"/>
      ...
      <xsl:template match="serif">
        <xsl:copy-of select="."/>
        <xsl:element name="lowercase">false</xsl:element>
      </xsl:template>
    </xsl:stylesheet>
  </do>
</personalizationRule>
<personalizationRule> <!-- - - - - - - - - reconciling type heterogeneity -->
 <on>
  //interaction[sender="http://www.productionDesigns.com"]
 </on>
 <if>
  //serif
 </if>
 <do>
  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
   <xsl:output method="xml" indent="yes"/>

   ...
   <xsl:template match="serif[text()='true']">
    <xsl:element name="serif">yes</xsl:element>
   </xsl:template>

   <xsl:template match="serif[text()='false']">
    <xsl:element name="serif">no</xsl:element>
   </xsl:template>

   <xsl:template match="serif[text()='1']">
    <xsl:element name="serif">yes</xsl:element>
   </xsl:template>

   <xsl:template match="serif[text()='0']">
    <xsl:element name="serif">no</xsl:element>
   </xsl:template>

  </xsl:stylesheet>
 </do>
</personalizationRule>
```

Fig. 5. Resolving Conflicts with Personalization Rules

by executing the XSLT program that captures it. The goal is to transform M into M'. Notice, for example, that there is no need to side-effect the host environment. Once the personalization stage is completed, M' is handed over to the classical Axis chains.

The resolution of heterogeneity conflicts such as described in Section 3 can be expressed using personalization rules, as the examples in Figure 5 illustrates.

One of the advantages of the approach proposed in this paper is that personalization engines can be deployed at the SOAP engines of either (or both) the interacting parties. Assume now that the personalization framework is deployed at ET and affects the arrival of requests that are sent to its services. Thus, when the request for a typeface arrives from PDP in the form of a SOAP message, the personalization engine identifies this and stamps the message appropriately. Then, the engine retrieves the rules pertaining to ET whose event is either `true` or an XPATH expression that matches the header of the SOAP message. Assume that the header of the SOAP message flowing from PDP to ET is such as to cause the XPath expressions on the event part of the rules in Figure 5 to be triggered.

If the condition of a triggered rule is either `true` or an XPath expression that matches its target, then it will fire. Figure 2 shows that the PDP message requesting ET to design a typeface will contain a `varieties` node. Thus, the XPath expression in the condition part of the personalization rule in Figure 5 for reconciling name heterogeneity will match and the rule will fire. The personalization rule for reconciling cardinality heterogeneity will also fire, because the SOAP message flowing from PDP to ET will not contain a `lowercase` node. Finally, the personalization rule in Figure 5 for reconciling type heterogeneity will also fire because of the match with a `serif` node.

Since the rules in Figure 5 have all fired, the XSLT programs in their action parts are executed. The first personalization rule in Figure 5 maps the `varieties` node in the original message M into a `styles` node in M', the personalized one, thereby resolving the name conflict between PDP and ET. The second personalization rule resolves the cardinality conflict by introducing, just below the `serif` node, a `lowercase` node and setting its value to `false` which is the default specified in the WSDL document for the service. Finally, the third personalization rule in Figure 5 resolves the type conflict by changing the type of the `serif` node, which is of type `boolean` in PDP to the ET-defined type `choices`. Thus, `'true'` and `1` (which are the representations for a true value in XML Schema) are mapped to `yes`, whereas `'false'` and `0` are mapped to `no`.

5 Related Work

The approach described in this paper is best understood in the context of research on web service composition, where the resolution of heterogeneities between web services becomes a focus of attention.

eFlow [4] is a platform to specify, enact and monitor e-service composition. Among other features, it can personalize process instances to a specific instance (or set of instances). Personalization relies on condition-action rules to modify

the schema of the process at run time, relying for that on the users' understanding of the process schema. This raises significant risks, not incurred in our approach. Our approach uses personalization as a way to cope dynamically with changes in services and maximize business opportunities, whereas eFlow uses personalization to cope with specific and exceptional situations. It is doubtful whether eFlow would be sufficiently agile and lightweight to underpin opportunistic VEs as targeted by our approach.

The approach in [16] uses a mediator to reconcile interfaces provided by a server with those assumed by a client. It suggests that this process could be applied in the context of web services using XSLT to effect transformations in SOAP messages flowing between web services, as is done in this paper. However, the infrastructure sketched in [16] is, in comparison, very large and very heavy. Also, to the best of our knowledge no precise, detailed, web service-based design, let alone an implementation, exists of it. It is doubtful whether the approach in [16] would be sufficiently agile and lightweight to underpin opportunistic VEs as targeted by our approach.

WebTransact [15] is a framework to enable web service composition using mediator services. These mediator services are wrappers which aim to provide applications with a homogeneous view of heterogeneous web services. Each mediator service is associated with one or more remote service and maps the web service description (provided in WSDL and WSTL, their homegrown web service transaction language) to the target form. While our approach expresses mediation as restricted, well-behaved ECA rules, WebTransact requires much more unrestricted wrappers (and hence more error-prone and more costly to develop and deploy). Also, while our approach uses no non-standard formalism (since XML, XPATH, XSL, WSDL and SOAP are W3C-sponsored and no dependency on BPEL4WS exists), WebTransact uses WSTL, which is homegrown. Finally, note that our approach deploys the mediation component inside the SOAP engine and hence implies minimal effort and minimal disruption, whereas WebTransact must deploy its wrappers visibly (and hence, comparatively obtrusively).

6 Conclusions

This paper has taken inspiration on past work on federating databases to address the issue of conflict resolution in web service federations and constitutes an evolution and an application of our recent work on personalization [2] and web service federation [1]. The paper has contributed a lightweight, unobtrusive approach to the resolution of such conflicts. The personalization framework described here allows either the server to adjust to many clients or a client to adjust to many servers (or both) without code maintenance to the web services themselves. The availability of this mechanism makes web service federations more suited still for modelling dynamic, opportunistic VEs. The approach has been implemented as described and the first evaluation results confirm the intuition that our contributions constitute an enabling technology to facilitate the formation and operation of agile, highly dynamic, opportunistic VEs.

References

1. M. A. T. Aragão and A. A. A. Fernandes. Characterizing Web Service Substitutivity with Combined Deductive and Inductive Engines. In *Advances in Information Systems, Second International Conference, ADVIS 2002*, pages 244–254, 2002.
2. V. R. Aragão, A. A. A. Fernandes, and C. A. Goble. Towards an Architecture for Personalization and Adaptivity in the Semantic Web. In *Third International Conference on Information Integration and Web-Based Applications and Services*, pages 139–149, 2001.
3. M. W. Bright, A. R. Hurson, and S. H. Pakzad. A Taxonomy and Current Issues in Multidatabases Systems. *IEEE Computer*, 25(3): 50–60, 1992.
4. F. Casati, S. Ilnicki, L. Jin, V. Krishnamoorthy, and M.-C. Shan. Adaptive and Dynamic Service Composition in eFlow. Technical report, HP Laboratories Palo Alto, 2000.
5. W. W. W. Consortium. W3c technical reports and publications. `http://www.w3c.org/TR/`.
6. T. A. S. Foundation. Apache xindice. http://xml.apache.org/xindice/.
7. T. A. S. Foundation. Axis. http://xml.apache.org/axis/.
8. K. Gottschalk, S. Graham, H. Kreger, and J. Snell. Introduction to Web Services Architecture. *IBM Systems Journal*, 41(2): 170–177, 2002.
9. W. Kim, I. Choi, S. Gala, and M. Scheevel. On Resolving Schematic Heterogeneity in Multidatabase Systems. *Distributed and Parallel Databases*, 1(3): 251–279, 1993.
10. W. Kim and J. Seo. Classifying Schematic and Data Heterogeneity in Multidatabase Systems. *IEEE Computer*, 24(12): 12–18, 1991.
11. S. N. Laboratories. Jess. http://herzberg.ca.sandia.gov/jess/.
12. A. Y. Levy. Logic-Based Techniques in Data Integration . In J. Minker, editor, *Logic-Based Artificial Intelligence*. Kluwer, 2000.
13. N. Nayak, K. Bhaskaran, and R. Das. Virtual Enterprises – Building Blocks for Dynamic e-Business . In *Proc. IT for Virtual Enterprises*, pages 80–87. IEEE Press, 2001.
14. N. W. Paton, editor. *Active Rules in Database Systems*. Springer, 1998. ISBN 0-387-985298.
15. P. F. Pires, M. Benevides, and M. Mattoso. Building Reliable Web Services Compositions. In *International Workshop Web Services Research, Standardization, and Deployment*, pages 551–562, 2002.
16. G. Smith. Component Adaptation of Web Services. In *13th Australian Software Engineering Conference*, Canberra, Australia, 2001.
17. A. Umar and P. Missier. A Framework for Analyzing Virtual Enterprise Infrastructure. In *Proc. RIDE-VE'99 Workshop on IT for Virtual Enterprises*, pages 4–11. IEEE Press, 1999.
18. WebServices.Org. `http://www.webservices.org/`.
19. G. Wiederhold. Mediators in the Architecture of Future Information Systems. *IEEE Computer*, 25(3): 38–49, 1992.
20. G. Wiederhold and M. Genesereth. The Conceptual Basis for Mediation Services. *IEEE Expert*, 12(5): 38–47, 1997.