

Building Reliable Web Services Compositions

Paulo F. Pires, Mário R. F. Benevides, and Marta Mattoso

Computer Science Department
COPPE - Federal University of Rio de Janeiro
P.O. Box 68511, Rio de Janeiro, RJ, 21945-970, Brazil
Fax: +55 21 22906626
{ pires, mario, marta } @cos.ufrj.br

Abstract. The recent evolution of internet technologies, mainly guided by the Extensible Markup Language (XML) and its related technologies, are extending the role of the World Wide Web from information interaction to service interaction. This next wave of the internet era is being driven by a concept named *Web services*. The Web services technology provides the underpinning to a new business opportunity, i.e., the possibility of providing value-added Web services. However, the building of value-added services on this new environment is not a trivial task. Due to the many singularities of the Web service environment, such as the inherent structural and behavioral heterogeneity of Web services, as well as their strict autonomy, it is not possible to rely on the current models and solutions to build and coordinate compositions of Web services. In this paper, we present a framework for building reliable Web service compositions on top of heterogeneous and autonomous Web services.

1 Introduction

Web services can be defined as modular programs, generally independent and self-describing, that can be discovered and invoked across the Internet or an enterprise intranet. Web services are typically built with XML, SOAP, WSDL, and UDDI specifications [19]. Today, the majority of the software companies are implementing tools based on these new standards [7, 10]. Considering how fast implementations of these standards are becoming available, along with the strong commitment of several important software companies, we believe that they will soon be as widely implemented as HTML is today.

According to the scenario just described, an increasing number of on-line services will be published in the Web during the next years. As these services become available in the Web service environment, a new business opportunity is created, i.e., the possibility of providing value-added Web services. Such value-added Web services can be built through the integration and composition of basic Web services available on the Web [3].

Web service composition is the ability of one business to provide value-added services to its customers through the composition of basic Web services, possibly offered by different companies [3, 4]. Web service composition shares many

requirements with business process management [1]. They both need to coordinate the sequence of service invocation within a composition, to manage the data flow between services, and to manage execution of compositions as transaction units. In addition, they need to provide high availability, reliability, and scalability. However, the task of building Web service compositions is much more difficult due to the degree of *autonomy*, and *heterogeneity* of Web services. Unlike components of traditional business process, Web services are typically provided by different organizations and they were designed not to be dependent of any collective computing entity. Since each organization has its own business rules, Web services must be treated as strictly autonomous units. Heterogeneity manifests itself through structural and semantic differences that may occur between semantically equivalent Web services. In a Web service environment it is likely to be found many different Web services offering the same semantic functionality thus, the task of building compositions has to, somehow, deal with this problem.

This paper introduces a framework, named *WebTransact*, which provides the necessary infrastructure for building reliable Web service compositions. The WebTransact framework is composed of a multilayered architecture, an XML-based language (named *Web Services Transaction Language*), and a transaction model. The multilayered architecture of WebTransact is the main focus of this paper. A detailed discussion on all other components of WebTransact can be found in [11].

The remainder of this paper is organized as follows. In Section 2, we start presenting a general picture of the WebTransact architecture. Next, we explain the components of that architecture. In Section 3, we discuss the related work. Finally, in Section 4, we present our concluding remarks.

2 The WebTransact Architecture

As shown in Fig. 1, WebTransact¹ enables Web service composition by adopting a multilayered architecture of several specialized components [12]. Application programs interact with *composite mediator services* written by composition developers. Such compositions are defined through transaction interaction patterns of *mediator services*. Mediator services provide a homogenized interface of (several) semantically equivalent *remote services*. Remote services integrate Web services providing the necessary mapping information to convert messages from the particular format of the Web service to the mediator format.

The WebTransact architecture encapsulates the message format, content, and transaction support of multiple Web services and provides different levels of value-added services. First, the WebTransact architecture provides the functionality of uniform access to multiple Web services. Remote services resolve conflicts involving the dissimilar semantics and message formats from different Web services, and conflicts due to the mismatch in the content capability of each Web service². Besides resolving structural and content conflicts, remote services also provide information on the interface and the transaction semantics supported by Web services. Second,

¹ The architecture of WebTransact is based on the Mediator technology [20].

² These conflicts are better explained in Section 2.3.

Mediator services integrate semantically equivalent remote services providing a homogenized view on heterogeneous Web services. Finally, transaction interaction patterns are built on top of those mediator services generating composite mediator services that can be used by application programs or exposed as new complex Web services.

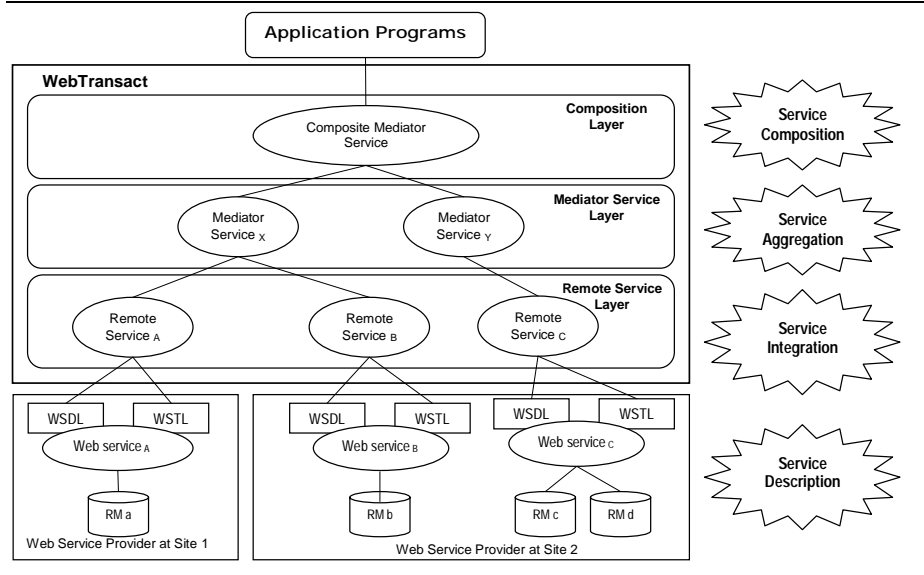


Fig. 1. The multilayered architecture of WebTransact.

WebTransact integrates Web services through two XML-based languages: Web Service Description Language (WSDL) [19], which is the current standard for describing Web service interfaces, and Web Service Transaction Language (WSTL) [11], which is our proposal for enabling the transactional composition of heterogeneous Web services. WSTL is built on top of WSDL extending it with functionalities for enabling the composition of Web services. Through WSDL, a remote service understands how to interact with a Web service. Through WSTL, a remote service knows the transaction support of the Web service. Besides the description of the transaction support of Web services, WSTL is also used to specify other mediator related tasks such as: the specification of mapping information for resolving representation and content dissimilarities, the definition of mediator service interfaces, and the specification of transactional interaction patterns of Web service compositions.

The distributed architecture of WebTransact separates the task of aggregating and homogenizing heterogeneous Web services from the task of specifying transaction interaction patterns, thus providing a general mechanism to deal with the complexity introduced by a large number of Web services.

The design of the WebTransact framework provides novel special features for dealing with the problems of Web service composition. Since mediator services provide a homogenized view of Web services, the composition developer does not have to deal with the heterogeneity nor the distribution of Web services. Another

important aspect of the WebTransact architecture is the explicit definition of transaction semantics. Since Web services describe their transaction support through WSTL definition, reliable interaction patterns can be specified through mediator service compositions.

2.1 The Remote Service Layer

Each remote service in WebTransact is a logical unit of work that performs a set of *remote operations* at a particular site. Besides its signature, a remote operation has a well-defined *transaction behavior*. The transaction behavior defines the level of transaction support that a given Web service exposes. There are two levels of transaction support. The first level consists of Web services that cannot be cancelled after being submitted for execution. Therefore, after the execution of such Web service, it will either commit or abort, and if it commits, its effects cannot be undone. The second level consists of Web services that can be aborted or compensated. There are two traditional ways to abort or compensate a previous executed service. One way, named *two-phase commit* (2PC) [8], is based on the idea that no constituent transaction is allowed to commit unless they are all able to commit. Another way, called *compensation*, is based on the idea that a constituent transaction is always allowed to commit, but its effect can be cancelled after it has committed by executing a compensating transaction. In order to accommodate these levels of transaction support, the WebTransact framework defines four types of transaction behavior of remote services, which are: *compensable*, *virtual-compensable*, *retriable*, or *pivot*. A remote operation is *compensable* if, after its execution, its effects can be undone by the execution of another remote operation. Therefore, for each compensable remote operation, it must be specified which remote operation has to be executed in order to undo its effects. The *virtual-compensable* remote operation represents all remote operations whose underlying system supports the standard 2PC protocol. These services are treated like compensable services, but, actually, their effects are not compensated by the execution of another service, instead, they wait in the prepare-to-commit state until the composition reaches a state in which it is safe to commit the remote operation. Therefore, virtual-compensable remote operations reference Web service operations whose underlying system provides (and exposes) some sort of distributed transaction coordination. A remote operation is *retriable*, if it is guaranteed that it will succeed after a finite set of repeated executions. A remote operation is *pivot*, if it is neither retriable nor compensable. For example, consider a simple service for buying flight tickets. Consider that such service has three operations, one, `reservation`, for making a flight reservation, another, `cancelReserv`, for canceling a reservation, and another, `purchase`, for buying a ticket. The `reservation` operation is compensable since there is an operation for canceling reservations. The `cancelReserv` operation is retriable since it eventually succeeds after a finite set of retries. On the other hand, the `purchase` operation is pivot because it cannot be undone (supposing non refundable ticket purchases).

In WebTransact, the concept of compensable, virtual-compensable, and pivot operation is used to guarantee safe termination of compositions. Due to the existence of dissimilar transaction behavior, some Web service providers may not support

transaction functionalities like the two-phase commit interface. In this case, it is not possible to put a remote operation in a wait state like the prepared-to-commit state. Therefore, after a pivot service has been executed, its effects are made persistent and, as there is no compensating service for pivot services, it is not possible to compensate its persistent effects. For that reason, pivot remote operations can be executed only when, after its execution, the composition reaches a state where it is ready to successfully commit. At this state, the system has guaranteed that there will be no need to undo any already committed service.

WSTL Document Example. The following example (Fig. 2, Fig. 3, and Fig. 4) shows a WSDL definition of a simple service providing car reservations, extended by the WSTL framework. The car reservation service supports two operations: reservation and cancelReservation.

The reservation operation returns a reservation code, or an error, in the parameter reservationResult of type string. The cancelReservation operation has only one parameter: reservationCode of type string. A valid value for this parameter is the value returned in a previous successfully executed request of operation reservation. The cancelReservation operation returns a value of type string indicating the success or failure of the executed request.

The definition of the car reservation service is separated in three documents³: data type definitions (Fig. 2), abstract definitions (Fig. 3), and transaction behavior definitions (Fig. 4).

```
<definitions targetNamespace="http://example.com/carReservation/Schema/"
...
  xmlns:tns="http://example.com/carReservation/Schema/"
  <types>
    ...
    <xsd:element name="reservationResponse">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="reservationResult" type="xsd:string"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="cancelReservation">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="reservationCode" type="xsd:string"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="cancelReservationResponse">
      ...
      <xsd:element name="string" type="xsd:string"/>
    </xsd:schema>
  </types>
```

³ Since the concrete WSDL definitions are not used for explaining the WSTL usage, they are not shown.

```
</definitions>
```

Fig. 2. Types definition for the car reservation service.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions
targetNamespace="http://example.com/carReservation/abstractDef/"
...
xmlns:lxsd="http://example.com/carReservation/Schema/"
...
<message name="reservationSoapIn">
  <part name="parameters" element="lxsd:reservation"/>
</message>
<message name="reservationSoapOut">
  <part name="parameters" element="lxsd:reservationResponse"/>
</message>
<message name="cancelReservationSoapIn">
  <part name="parameters" element="lxsd:cancelReservation"/>
</message>
<message name="cancelReservationSoapOut">
  <part name="parameters" element="lxsd:cancelReservationResponse"/>
</message>
<portType name="reservationSoap">
  <operation name="reservation">
    <input message="tns:reservationSoapIn"/>
    <output message="tns:reservationSoapOut"/>
  </operation>
  <operation name="cancelReservation">
    <input message="tns:cancelReservationSoapIn"/>
    <output message="tns:cancelReservationSoapOut"/>
  </operation>
</portType>
</definitions>
```

Fig. 3. Abstract definitions for the car reservation service.

```
<definition ...
xmlns:absd="http://example.com/carReservation/abstractDef/">
<wstl:transactionDefinitions>
  <wstl:transactionBehavior operationName=" absd:reservation"
    type="compensable">
    <wstl:activeAction portTypeName="svc:reservationSoap"
      compensatoryOper="cancelReservation">
      <wstl:paramLink>
        <wstl:sourceParamLink msgName="reservationSoapOut"
          param="absd:reservationResponse/@reservationResult"/>
        <wstl:targetParamLink msgName="cancelReservationSoapIn"
          param="absd:cancelReservation/@reservationCode"/>
      </wstl:paramLink>
    </wstl:activeAction>
  </wstl:transactionBehavior>
  <wstl:transactionBehavior operationName=" absd:cancelReservation"
    type="retriable"/>
</wstl:transactionDefinitions>
```

```
</definitions>
```

Fig. 4. Transaction behavior definitions for the car reservation service.

Considering the WSDL concept of abstract and concrete definitions, the transaction behavior describes the transaction semantics of *abstract operations* of Web services. The transaction behavior is a semantic concept related to operations, thus it is independent of network deployment or data format bindings of concrete endpoints and messages. Therefore, the transaction behavior should be inserted as a child element of a port type operation that describes the abstract portion of message exchanges. However, WSDL does not allow extensibility elements inside port type operations. The only WSDL element that supports extensibility and is located in the context of the abstract portion of a WSDL document is the definitions element. For this reason, WSTL defines its root element, `transactionDefinitions`⁴, as a direct child of the `wSDL:definitions` element.

In the car reservation service (Fig. 4), the `transactionDefinitions` element has two child `transactionBehavior` elements each containing transaction semantics information on the operations supported by that Web service.

The first `transactionBehavior` element has “`absd:reservation`” as the value of attribute `operationName`. The prefix “`absd:`” references the namespace “`http://example.com/carReservation/abstractDef/`”, which is the namespace for the abstract definitions of the car reservation service. The value “`absd:reservation`” is a QNAME that references the `wSDL:operation` element named `reservation`, which is defined in the WSDL document of Fig. 3. Therefore, the first `transactionBehavior` element in Fig. 4 defines the transaction behavior of the `reservation` operation of the car reservation service. The value “`compensable`” of attribute `type` indicates that `reservation` operation is *compensable*, as defined in Section 2.1. The compensatory operation of the compensable operation `reservation` is defined by the `activeAction` element that has `absd:reservationSoap` and `absd:cancelReservation` as the values of attributes `portTypeName` and `compensatoryOper`, respectively. These values define the `cancelReservation` operation of port type `reservationSoap` from the car reservation service (prefix `absd:`) as the compensatory operation for the `reservation` operation.

The `paramLink` element, which is a child element of the `activeAction` element, represents a data link involving a message part of the compensable operation `reservation` to a message part of its compensatory operation `cancelReservation`. This data link specifies a data flow from the compensable operation `reservation` to its compensatory operation `cancelReservation`, prescribing how the input parameters of the operation `cancelReservation` is constructed from the output parameters of the `reservation` operation. The `paramLink` element contains two child elements: the `sourceParamLink` and `targetParamLink` elements. The `sourceParamLink` element defines the origin of the data flow, while the `targetParamLink` element defines the destination of the

⁴ The complete WSTL specification can be found in [11].

data flow. In the example, the `sourceParamLink` element has `absd:reservationSoapOut` as its `msgName` attribute and the XPath expression `absd:reservationResponse/@reservationResult` as the value for its `param` attribute, while the `targetParamLink` element has `absd:cancelReservationSoapIn` as its `msgName` attribute and the XPath expression `absd:cancelReservation/@reservationCode` as the value for its `param` attribute. These elements link the result value of the operation `reservation` must to the input parameter of the operation `cancelReservation`.

The data link defined above is used by `WebTransact` to construct the input message of the compensatory operation `cancelReservation` when invoking it to compensate the work done by the operation `reservation` during an execution of a given transaction.

The second `transactionBehavior` element has “`absd:cancelReservation`” as the value of attribute `operationName`. This value indicates that this `transactionBehavior` element defines the transaction behavior of the `cancelReservation` operation of the car reservation service. The value “`reliable`” of attribute `type` indicates that the `cancelReservation` operation is *reliable*, as defined in Section 2.1. Note that there is no child element for this `transactionBehavior` element. The reason is that reliable operations are not compensable, thus there is no other necessary information on the operation transaction behavior to be provided by the Web service. Only *compensable* and *virtual-compensable* operations need further information besides the information available in the set of attributes of the `transactionBehavior` element.

In this section, we have only shown a simple example of the WSTL elements for describing compensable and reliable Web service operations. Other examples, including examples of virtual-compensable operations, can be found in [11].

2.2 The Mediator Service Layer

Mediator services aggregate semantically equivalent remote services, thus providing a homogenized view of heterogeneous remote services. Semantically equivalent remote services are services that integrate Web services exposing different WSDL interfaces but providing the same semantic functionality. Unlike remote services, which are logical units of work that perform remote operations at a particular site, mediator services are *virtual* services responsible for delegating its operations’ execution to one or more remote services. This delegation is done over a set of semantically equivalent remote services aggregated by the mediator service. Like remote operations, mediator service operations have a signature and a well-defined *transaction behavior*, which can be either *compensable*, *reliable*, or *pivot*.

The transaction behavior of one mediator service operation is based on the transaction behavior of its aggregated remote operations. If all aggregated remote operations have the same type of transaction behavior, e.g. *compensable*, then the transaction behavior of mediator service operation will have the same value, i.e., *compensable*. On the other hand, if the mediator service operation aggregates remote operations with different transaction behaviors, then its transaction behavior will be

the *least restrictive* transaction behavior among the transaction behaviors of its aggregated remote operations. The most restrictive transaction behavior is the pivot, followed by the retrievable transaction behavior, while both the compensable and virtual-compensable transaction behaviors are least restrictive transaction behaviors. The concept of least/most restrictive transaction behavior defines whether a mediator service operation can participate in a given composition execution. A mediator service operation, which aggregates at least one remote operation that is compensable (or virtual-compensable), can participate in any composition execution. On the other hand, a mediator service operation that aggregates only pivot (or retrievable) remote operations can participate only in compositions that call this mediator service operation after it reaches a state where it is ready to successfully commit. Recall from Section 2.1 that after a pivot remote operation has been executed, the composition has to enter a state where it is ready to successfully terminate. Thus, it is guaranteed that there will be no need to undo any already submitted pivot (or retrievable) remote operation. Therefore, mediator service operations that aggregate only pivot (or retrievable) remote service operations can only participate in compositions that call this mediator service operation when it is ready to successfully terminate.

Mediator service operations expose the same types of transaction behavior as remote service operations, except for the absence of the virtual-compensable operation. The specific transaction behavior of virtual-compensable remote operations is isolated by the mediator service operation that aggregates them. Mediator service operations that aggregate virtual-compensable remote operations expose the same interface of mediator service operations that aggregate only real compensable remote operations. This simplifies the protocol that coordinates the composition execution, since both compensable and virtual-compensable remote service operations are treated, at the composition level, as a single operation type.

2.3 Resolving Semantic and Content Dissimilarities of Web Services

In order to provide a homogenized layer of services, each mediator service exposes a single interface that is used by composition specifications. As mediator services aggregate semantically equivalent remote services, which possibly have different interfaces, it is necessary to provide mapping information between the interface supported by the mediator service and each one of the interfaces supported by its aggregated remote services. In WebTransact, each WSDL port type is imported as a new remote service. Since the WSDL port type element defines the syntax for calling a set of remote operations, i.e., a specific supported interface, each WSDL port type definition is considered as a separated remote service. A remote service links a mediator service to a WSDL port type element and it provides *mapping information* between mediator service operations and port type operations and specifies the *content description* of the remote service. The mapping information prescribes how the input parameters of a remote service operation are constructed from the input parameters of its related mediator service operation, as well as how the output parameters or fault messages received from that remote service operation are mapped to the output or fault messages of its related mediator service operation. The mapping information also defines the *transaction outcome* of a given remote service operation. In order to define messages signaling an unsuccessful terminating state, all that is needed is the definition of mapping information linking fault messages and/or specific

return value of output messages of a remote service operation to a fault message of a mediator service operation. The content description specifies whether a remote service is able to execute a particular service invocation. For example, consider remote services r_{m_1} and r_{m_2} providing car reservations. Remote service r_{m_1} can make car reservations world wide, while remote service r_{m_2} accepts only car reservations in Brazil. Now, consider that mediator service ms_1 aggregates r_{m_1} and r_{m_2} . If ms_1 receives a request to make a car reservation inside USA then, ms_1 will invoke only the remote service r_{m_1} . The mediator service ms_1 knows, using the remote service content description, that r_{m_2} is not able to make car reservation outside Brazil. Mediator services and remote services are both specified through WSTL definitions.

2.4 The Composition Layer

A composite mediator service describes transaction interaction patterns from a set of cooperating mediator service operations necessary to accomplish a task. Such interaction pattern defines the execution sequence of mediator service operations as well as the level of atomicity and reliability of a given composition. In WebTransact, a composition is specified using WSTL elements. The WSTL elements for specifying composite mediator service as well as their operational semantics are described in [11].

WSTL models compositions as *composite tasks*. A composite task is represented by a labeled directed graph in which nodes represent steps of execution and edges represent the flow of control and data among different steps. Each step of a composite task is either an atomic *task* or another composite task. An atomic task is a unit of work that is executed by a mediator service operation. Therefore, atomic tasks have a mediator service operation assigned to it, which is invoked when the task is executed.

Tasks are identified by a name and have a signature, a set of execution dependencies, a set of data links, and, optionally, a set of rules.

The *signature* of an atomic task is related to the input, output, and fault messages of the mediator service operation that is used as the implementation of the task. The *signature* of a composite task is related to the input, output, and fault messages of the component tasks used as the implementation of the task. A component task can be an atomic task or another composite task.

Execution dependencies are based on the execution state of component tasks defining the first kind of edges in the graph that represent a composite task. An execution dependency is defined among related component tasks and it is a constraint on the temporal occurrence of the start and termination events of them. Execution dependencies define the order in which tasks must be executed, i.e., the composition control flow. An execution dependency is always specified based on the *execution state* of one component task.

Data links are mappings between messages belonging to the signatures of related tasks to allow the exchange of information between these tasks. Data links are the second kind of edges in the graph that represents a composite task.

Rules specify the conditions under which certain events will happen. Rules can be associated to dependencies or to data links. A dependency that has a rule will evaluate to true if both the dependency *and* the rule evaluate to true. A data link that has a rule

will evaluate to true if the rule evaluates to true. Data links without explicit rules always evaluate to true.

Besides the components described above, composite tasks have a set of *mandatory tasks*. This set specifies the component tasks that must commit in order to commit the composite task. A user can specify a composite task aggregating tasks that are desirable, but not essential, to accomplish the target task. The set of mandatory tasks allow the distinction between *desirable* and *mandatory* tasks, providing more flexibility while specifying a composition. This flexibility increases the composition robustness, since the set of tasks required to commit, in order to commit the composition, are formed only by that tasks that are essential to accomplish the composition target. Thus, the composition will successfully terminate even if a subset of its component tasks fails, as long as all its mandatory tasks successfully commit.

3 Related Work

The WebTransact framework is a multidisciplinary work that is related with many other areas such as e-service composition, transactional process coordination, workflow management systems, and distributed computing systems.

There has been extensive research in transaction support for distributed computing systems [5, 14], transactional process coordination [2, 15], and in workflow management systems [6]. While these projects address the support of distributed transactions, they do not consider the coordination of service capabilities of autonomous remote service providers. Since Web services support dissimilar capabilities with respect to its transaction behavior, this is a significant difference. Thus, implementing transaction semantics across the Web services becomes much more difficult when compared to a scenario where all distributed components support identical transaction behavior. Moreover, these works were conceived before the current stage of the World Wide Web. Therefore, they do not consider the XML-based standards that enable the Web service technology.

The existent works in the area of e-service composition (WSFL [9], XLANG [16], and WSCL [18]) are concentrated in defining primitives for composing services and automating service coordination. The majority of these works consider XML-based standards for Web service technology. However, the primitives for composition proposed by these works do not directly address the problems associated with the necessary *homogenization* of Web services. In addition, the transaction support proposed in this area does not consider the coordination and mediation of Web services with dissimilar transaction support.

More recently, some specific transaction protocols for the Web have being discussed, such as the W3C tentative hold protocol (THP) [17], and the OASIS Business Transaction Protocol (BTP) [13]. These protocols define a model for coordinating the transaction execution of Web services based on a predefined set of transaction messages. Still, they propose a transaction model based on a relaxed notion of the all-or-nothing property of conventional transactions. While these works provide support for distributed transactions on the Web environment through

enforcing a predefined set of transaction messages, WSTL provides a mean of coordinating distributed transactions on the Web without enforcing all participants (Web services) to support a unique and standard transaction protocol. In this sense, WSTL provides more flexibility while preserving the autonomy of Web services.

4 Conclusions

The Web services technology provides the underpinnings to a new business opportunity where one company can offer value-added services to its customers through the composition of basic Web services. However, the current Web services technology solves only part of the problem of building Web services compositions. Building reliable Web service compositions requires much more than just addressing interoperability between client programs and Web services. Besides interoperability, building Web services compositions requires mechanisms for: describing the dissimilar transaction support of Web services, resolving the semantic and content heterogeneity of semantically equivalent Web services, specifying the transaction interaction patterns among Web services, and coordinating such interaction patterns. Still, due to this novel set of requirements, posed by the Web services environment, existent business process frameworks cannot be directly applied to develop Web service compositions. Therefore, there is a need of new frameworks, specifically developed for addressing the new requirements of the Web service environment. In this paper, we have introduced one of such frameworks, the WebTransact framework.

WebTransact treats the problem of building composition in an integrated way, providing mechanisms for describing the dissimilar transaction behavior of Web services, for aggregating semantically equivalent Web services and for resolving their heterogeneities, for specifying reliable interaction patterns of Web services, and for coordinating such interaction patterns in a transactional way. To the best of our knowledge, there is no other works on integrated frameworks for Web service composition addressing the requirements encompassed by WebTransact.

References

1. Alonso, G., Fiedler, U., Hagen, C., et al., "WISE: Business to Business E-Commerce". In: Proceedings of RIDE, Sydney, Australia, 1999.
2. Alonso, G., Schuldt, H., Schek, H., "Concurrency Control and Recovery in Transactional Process Management". In: *Proceedings of the Symposium on Principles of Database Systems in Philadelphia*, pp. 316-26, 1999.
3. Casati, F., Ilnicki, S., Jin, L., et al., "Adaptive and Dynamic Service Composition in eFlow". In: Proceedings of CaiSE 2000, Stockholm, Sweden, pp. 13-31, June 2000.
4. Casati, F., Shan, M., "Dynamic and adaptive composition of e-services", *Information Systems*, v. 26, n. 3, pp. 143-163, 2001.
5. Elmagarmid, A. K., *Database Transaction Models for Advanced Applications*. Morgan Kaufmann, 1992.

6. Georgakopoulos, D., Hornick, M., Sheth, A., "An overview of workflow management: from process modeling to workflow automation infrastructure", *Intl. Journal on distributed and parallel databases*, v. 3, n. 2 , pp. 119-153, 1995.
7. IBM White Paper, "The IBM WebSphere software platform and patterns for e-business - invaluable tools for IT architects of the new economy". [<http://www4.ibm.com/software/info/websphere/docs/wswhitepaper.pdf>], 2000.
8. Lampson, B. W., "Atomic Transactions". In: Goos, G., Hartmanis, J. (eds.), *Distributed Systems - Architecture and Implementation: An Advanced course*, Springer-Verlag, pp. 246-265, 1981
9. Leymann, F., "Web Services Flow Language (WSFL 1.0)". [<http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf>], May 2001.
10. Microsoft White Paper, "A Blueprint for Building Web Sites Using the Microsoft Windows DNA Platform". [<http://www.microsoft.com/commerceserver/techres/whitepapers.asp>], 2000.
11. Pires, P. F., Benevides, R. F. M., Mattoso, M., "WebTransact: A Framework for Specifying and Coordinating Reliable Web Service Compositions". In: Technical Report ES-578/02 PESC/Coppe Federal University of Rio de Janeiro, [<http://www.cos.ufrj.br/~pires/webTransact.html>], April, 2002.
12. Pires, P.F., Raschid, L., "MedTransact: Transaction Support for Mediation with Remote Service Providers". In: *Proceedings of the 3rd International Conference on Telecommunications and Electronic Commerce*, Dallas, USA, November, 2000.
13. Potts, M., Cox, B., Pope, B., "Business Transaction Protocol Primer". OASIS Committee Supporting Document, [<https://www.oasis-open.org/committees/business-transactions/documents/primer/Primerhtml/BTP%20Primer%20D1%2020020602.html>], June 2002.
14. Ramamritham, K., Chrysanthis, P. K., ed., *Advances in Concurrency Control and Transaction Processing*, IEEE Computer Society Press, CA, 1997.
15. Schudt, H., Schek, H. J., Alonso, G., "Transactional Coordination Agents for Composite Systems". In: *Proceedings of the International Database Engineering and Applications Symposium (IDEAS 1999)*, Montreal, Canada, pp. 321-331, August 1999.
16. Thatte, S., "XLANG: Web Services for Business Process Design". [http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm], Microsoft Corporation, 2001
17. W3C (World Wide Web Consortium) Note, "Tentative Hold Protocol Part 1: White Paper". [<http://www.w3.org/TR/tenthhold-1/>], November 2001.
18. W3C (World Wide Web Consortium) Note, "Web Services Conversation Language (WSCL) 1.0". [<http://www.w3.org/TR/2002/NOTE-wscl10-20020314/>], March 2001.
19. W3C (World Wide Web Consortium) Note, "Web Services Description Language (WSDL) 1.1". [<http://www.w3.org/TR/2001/NOTE-wsdl-20010315/>], March 2001.
20. Wiederhold, G., "Mediation in Information Systems", *ACM Computing Surveys*, v. 27, n. 2, pp. 265-267, 1995.