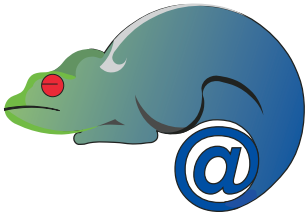
ADAPT
IST-2001-37126

Middleware Technologies for Adaptive and Composable Distributed Components

Evaluation Plan



Deliverable Identifier: D15
Delivery Date: 20th March 2003
Classification: Public Circulation
Authors: Stuart Wheater, Jim Webber, Mark Little
Document version: 1.0 20th March 2003

Contract Start Date: 1 September 2002
Duration: 36 months
Project coordinator: Universidad Politécnica de Madrid (Spain)
Partners: Universitá di Bologna (Italy), ETH Zürich (Switzerland), McGill University (Canada), Universitá degli Studi di Trieste (Italy), University of Newcastle (UK), Arjuna Technologies Ltd (UK)

Project funded by the European Commission under the Information Society Technology Programme of the 5th Framework (1998-2002)

Contents

1 ADAPT Evaluation Plan

The purpose of this document is to set out a plan for evaluating the results of the ADAPT project. This evaluation plan will be used to not only to evaluate the project but also to serve as a guide during the project.

1.1 Strategy for Evaluating Functional Capabilities and Attributes

The strategy for evaluating the results of the ADAPT project is to identify the key technical goals of the project, then based on these goals, to compare the technical capabilities and attributes of the ADAPT approach with those of a selected set of existing and proposed technologies. These comparisons leading to question which can by used to evaluate the results of the ADAPT project. The key technical goals of the ADAPT project have been identified as to make advances in support for:

- Availability
- Scalability
- Performance
- Service description
- Composability
- Configuration
- Adaptation
- Process definition
- Replication
- Security
- Transaction models

Many of these technical goals are related, such as composability and configuration, but this list covers the key technical goals of the ADAPT project.

The set of selected existing and proposed technologies that have been analyzed are:

- Java 2 Platform, Enterprise Edition (J2EE)
- Business Transaction Protocol (BTP)
- Web Services Transactions (WS-Transaction) and Web Services Coordination (WS-Coordination)
- Business Process Execution Language for Web Service (BPEL4WS)
- Web Service Choreography Interface (WSCI)
- Web Service Flow Language (WSFL)
- Microsoft's XLANG
- Web Service Conversation Language (WSCL)
- Web Service Description Language (WSDL)
- Web Service Security (WS-Security)
- Universal Description, Discovery and Integration (UDDI)
- OMG Interface Definition Language (OMG IDL)
- CORBA Interface Repository (CORBA IR)

In section 2 each of these technologies will be analyzed, and their functional capabilities and attributes which are relevant to the ADAPT project are identified, and questions which can be used to evaluate the functionality of the ADAPT project are derived.

1.2 Strategy for Evaluating Non-Functional Capabilities and Attributes

Another aspect of the evaluation strategy is to identify and evaluate non-functional capabilities and attributes. The main non-functional goals of the ADAPT project are in the areas of Availability, Scalability and Performance. To allow the evaluation of the Availability, Scalability and Performance of the ADAPT approach two series of applications will be constructed.

The first series of applications will be used to compare the ADAPT platform with other comparable J2EE platforms, including JBoss, the platform which has been augmented with features to form the ADAPT platform. This series of applications will be used to evaluate the Availability, Scalability and Performance of the ADAPT platform compared to other platforms. To allow this, the series of applications will be required to have the following characteristics:

- Simulate a realistic interaction pattern, between application and platform.
- Interact with all sub-systems modified as part of the ADAPT project.
- Expose an implementation that is based on standard APIs, so lessening the effort required in porting between different platforms.

This series of applications will be used to answer evaluation questions such as:

- In the case of partial failure of machines on which an application is running, does the ADAPT platform provide better availability that other platforms?
- As the number of clients of an application increases, does the ADAPT platform provide better performance that other platforms?
- As the number of machines in the “cluster” on which an application runs increases, does the ADAPT platform provide better performance that other platforms?

The second series of applications will be used to evaluate the feature enhancements which are unique to the ADAPT platform, in particular those for adaptability. The requirement of this series of applications will become clearer as the capabilities of the ADAPT platform are specified.

1.3 Evolution for the Evaluation Plan

It is intended that this evaluation plan will evolve during the lifetime of the project. In future versions of this document, additional goals for the projects may be identified, and additional technologies analysed. Four possible technologies that may be included in a future version of the evaluation plan are:

- Microsoft .Net [16]
- CORBA Component Model (CCM) [17]
- Java Management Extensions (JMX) [18]
- Grid Service [19] [20]

The evaluation plan will also be updated to reflect technical decisions in the Composite Service (CS) and Basic Service (BS) Architectures.

2 Analysis of Existing and Proposed Technologies

2.1 Java 2 Platform, Enterprise Edition (J2EE)

2.1.1 Summary

The Java 2 Platform, Enterprise Edition (J2EE) [2] is a collection a Java APIs that support commonly used distributed computing technologies and network services. The purpose of these APIs is to support the rapid development of Enterprise Applications. In general, the implementer of an Enterprise Application has many different issues to address, for example, providing Internet interfaces along with providing Intranet interfaces, also having to accessing legacy applications and databases.

A typical J2EE Enterprise Application provides its Internet interfaces via web pages, sometimes e-mail and in the future via Web services [1]. As for Intranet interfaces, J2EE supports both remote procedure call and message oriented integration with legacy applications and supports interaction with SQL based relational databases.

The J2EE APIs are numerous and provide an extensive set of functionality required to construct Enterprise Applications. In the rest of this section the following J2EE APIs will be briefly described:

- JDBC
- EJB
- Java IDL (CORBA)
- RMI
- JNDI
- JMS
- JTA
- Servlet/JSP
- JavaMail

The JDBC API allows a Java application to access relational databases, which may or may not be remotely hosted. Through the JDBC API, applications can execute SQL statements, retrieve results, and propagate changes back to a database. The designers of JDBC have attempted to create a platform-neutral interface between database and application, where database vendors or third-party developers providing a JDBC driver, which is a set of classes that implements the required functions for a particular database.

Enterprise Java Beans (EJB) is a component model for units of business logic and business data. The EJB model attempts to separate the beans, their container and the server. The beans are provided by the application implementer, and three types of bean are supported: session, entity and message-driven. Session beans are used to encapsulate business logic; Entity beans are used to encapsulate business data; Message-driven beans are used to support asynchronous interaction with business logic. The EJB container manages a set of beans, handling such thing as: lifecycle management, instance pooling, distributed transaction management and security. The EJB server itself hosts the EJB containers.

The Java IDL API provides an interface between Java application and distributed object and services build using the Common Object Request Broker Architecture (CORBA). It is possible that such distributed objects and services could be legacy applications.

The Java Remote Method Invocation (RMI) API is Java's native scheme for creating and using remote objects. Java RMI is "Java native" because it deals directly with Java objects, this makes Java RMI particularly suitable for accessing remote EJBs.

The Java Naming and Directory Interface (JNDI) is an API that supports accessing naming and directory services from Java applications. JNDI is an integral part of the J2EE framework; it is used by J2EE components to access various runtime resources such as the transaction manager, EJB home references and JDBC data sources.

The Java Messaging Service (JMS) provides an API for Java applications to perform reliable asynchronous messaging. The JMS API is a portable interface between Java applications and a native message-oriented middleware (MOM) system. This allows Java applications to interact with legacy applications through the MOM system.

The Java Transaction API (JTA) allows Java applications to manage ACID transactions based on the X/Open XA API for distributed transactions.

The Java Servlet API and JavaServer Pages (JSP) provide a standard way to extend Web servers to support dynamic content generation. These two techniques though related provide different approaches dynamic content generation. Java Servlet API allows operations to be written which directly handle HTTP requests, and directly generate content, usually HTML, as a response to the request. It is not unusual for this generated content to contain information that has been obtained via EJBs from a database. JSPs provide a higher-level approach; a JSP is usually HTML or XML that has embedded within it special tags that cause fragments of Java to be executed. These fragments of Java can cause HTML or XML to be generated that is specific to the request.

The JavaMail API provides a platform and protocol-independent framework to build Java applications that utilise Internet e-mail. This allows J2EE Enterprise Applications to interact with users or other programs via e-mail.

As well as defining APIs, the J2EE standard also addresses issues such as deployment. The structure of entities called WAR (Web Application aRchive) and EAR (Enterprise Application aRchive) files are specified. WAR files contain static content such as HTML files, along with JSP and the Java class required to generate dynamic content. WAR files also contain a deployment descriptor, in XML, which specifies things such as how to map a URL to a servlet and how to map a servlet to a Java class. EAR files contain a collection WAR and JAR file that make up an enterprise application, along with a deployment descriptor, in XML, which specifies such things as the "context root" associated with a WAR file. Most J2EE platforms support redeployment of EAR and WAR files, that is if an EAR or WAR needs to be updated, then all that is required is to replace the existing file. The change will be spotted by the J2EE platform and the old application will be replaced by the new version.

2.1.2 Relevant Capabilities and Attributes

- Container managed transactions
- Container managed security
- Deployment descriptors
- Use of JNDI for component discovery
- Support for dynamic application updates

2.1.3 Evaluation Questions

- Does the ADAPT transaction model preclude the use of container managed transactions, and is this a major disadvantage?
- Does the ADAPT security model preclude the use of container managed security, and is this a major disadvantage?
- In J2EE deployment descriptors are complicated and hard to manage, if ADAPT has deployment descriptors, are they simpler and easier to manage?
- In J2EE deployment descriptors are relatively static, if ADAPT has deployment descriptors, are they more dynamic, and is this a major advantage?
- Does ADAPT allow dynamic application updates?

2.2 Business Transaction Protocol (BTP)

2.2.1 Summary

The Business Transaction Protocol (BTP) [3] is an OASIS specification designed to provide reliable coordination of parties engaged in a business-level transaction. Ratified to committee specification level in May 2002, BTP has the backing of several large IT organizations including HP, BEA, Sun, and Oracle as well as a number of small and medium-sized vendors.

BTP provides a common understanding and a way to communicate levels of participation within transactions and limits on these levels between organizations. The formal rules are necessary for the distribution of parts of business processes outside the boundaries of an organization. BTP solves part of the problem for developers of loosely coupled transactions—the coordination of services/participants to ensure a consistent termination outcome. Expertise in the design of compensating actions is still required, but these compensations are local rather than distributed.

BTP uses a two-phase completion protocol for transaction coordination. At termination time, services participating in a transaction are asked up-front to state their intention (whether they will proceed with the transaction or whether they are not prepared to do so), and will later be instructed by the transaction manager to either proceed or not based on the analysis of all of the collected intentions. In order to satisfy its requirements, BTP supports two distinct transactions models, which are:

- Atoms: Similar to traditional atomic transactions where all Web services participating in an Atom are *guaranteed* to see the same outcome as all of the other participants: the outcome is atomic.

- Cohesions: Which allow business logic to dictate which combination of participating services succeed, while permitting the transaction as a whole to make forward progress – this allows the business transaction to proceed even in the presence of failures.

Cohesions allow us to pick groups of participating services (known as the *confirm set*) that we would eventually like to reach completion: unlike an atom, not all participants need see the same outcome for the transaction, i.e., atomicity is relaxed.

The BTP transaction coordinator is not as dictatorial as its equivalents in other transaction management models: the BTP model recognizes the possibility that other parts of an application might need to influence the decision making process required to complete a transaction. In addition, BTP supports runtime negotiation of quality of service characteristics based on the exchange of *qualifiers*. Qualifiers are the mechanism which enables the bilateral exchange of protocol "small print" between participants and coordinators. In essence, each BTP message allows the sender to tag qualifiers that describe such things as, "I will be prepared for the next ten minutes, and after that I will unilaterally cancel" and "You must be available for at least the next 24 hours to participate in this transaction." Qualifiers are a valuable mechanism in Web services transactions because in a loosely coupled environment, it is extremely useful to know that the party you're communicating with will only be around for so long, or to be able to specify that your party won't hang around while others procrastinate.

2.2.2 Relevant Capabilities and Attributes

- Support for Atoms (a.k.a. BTP Atomic Transactions)
- Support for Cohesions (a.k.a. Cohesive Business Transactions)
- Support for Qualifiers

2.2.3 Evaluation Questions

- Through cohesions BTP supports non-atomic transactions, does ADAPT provide support for non-atomic transactions?
- BTP does not address the security of the transaction protocol, how does ADAPT address security of the transaction protocol?
- Through qualifiers BTP's transaction participants can dynamically negotiate their relationship to the coordinator, does ADAPT's transaction model provide a corresponding capability?

2.3 Web Services Transaction (WS-Transaction) and Web Services Coordination (WS-Coordination)

2.3.1 Summary

The purpose of the combined Web Services Transaction (WS-Transaction) [4] and Web Services Coordination (WS-Coordination) [5] specifications is to provide a standard for conducting transactions over Web services. The WS-Transaction and WS-Coordination specifications were published by IBM, Microsoft and BEA in August 2002, and are intended to be a competitor to the Business Transaction Protocol (BTP) [3] specification.

The current WS-Transaction and WS-Coordination specifications are lacking details that would be required to construct interoperable implementation, which are based purely on these specifications. The authors of these specifications have stated that it is their intention to submit them to a standards body such as W3C or OASIS.

The WS-Coordination specification describes an extensible framework for providing protocols that coordinate the actions of distributed applications. The framework consists of two simple message-oriented request-response protocols for *Activation* and *Registration*. Being message-oriented the request and response are separated into two one-way invocations.

The *Activation protocol* specifies how an application can request a coordination context from an activation service, for a specified coordination type. The returned coordination context will contain, at least, a unique identifier for the context, the coordination type, and the endpoint address of the registration service for that context. This coordination context can be passed between applications so allowing other application to register with the registration service for the context.

The *Registration protocol* specifies how an application can request that a registration service includes the application as a participant in the protocol associated with the context. As part of the registration protocol the application sends to the registration service its participant endpoint address, in response the registration service responds with the context's protocol coordinator's endpoint address.

Building on WS-Coordination, the WS-Transaction specification describes two coordination types Atomic Transactions and Business Activities. Associated with the Atomic Transaction coordination type are five simple message-oriented protocols for *Completion*, *Completion with Acknowledgement*, *Phase Zero*, *Two-Phase Commit* and *Outcome Notification*. Associated with the Business Activity coordination type are two simple message-oriented protocols for *Business Agreement* and *Business Agreement with Complete*.

The Atomic Transaction protocols provide support for standard all-or-nothing ACID transactions. Through the *Completion* and *Completion with Acknowledgement* protocols participants can request the terminations, commit or abort, of a transaction. If "with acknowledgement" the participant must acknowledgement receipt of the final outcome before the corresponding coordinator can safely forget the transaction. The final outcome of a transaction can also be requested using the *Outcome Notification* protocol. If a participant wishes to participate in the transaction as a resource the *Phase Zero* and *Two-Phase Commit* protocols are available.

The Business Activity protocols provide support for handling long-lived activities and the desire to apply business logic to handle business exceptions. The protocols allow a coordinator to control a participant within a business activity, for example, the coordinator can request that the activity abandon its work in some appropriate way, or after completion the coordinator can request compensation be performed.

2.3.2 Relevant Capabilities and Attributes

- Separation of "transaction protocols" and "activation and registration protocols".

- Support for ACID transactions.
- Support for long-lived activates.
- Asynchronous message-oriented interaction model.

2.3.3 Evaluation Questions

- How does ADAPT support long-lived activities?
- Does ADAPT support message-oriented interaction with its transaction infrastructure?

2.4 Business Process Execution Language for Web Services (BPEL4WS)

2.4.1 Summary

The purpose of Business Process Execution Language for Web Service (BPEL4WS) [6] is to provide a standard for specifying business process behaviour and business process interactions, for applications composed from Web services. The BPEL4WS specification was published by IBM, Microsoft and BEA in July 2002, and is intended to supersede the XLANG [9] and WSFL [8] specification.

The BPEL4WS integration model is that business partners interact through peer-level conversations, using both synchronous and asynchronous messages. These conversations are carried out between the partners using specified sets of Web services. The conversations are coordinated within a partner by a business process.

From the business process perspective, the services provided by other partners and the services expected by partners are specified as a set of service links. The service links, an extension of WSDL [11], are used to model the peer-to-peer partner relationships. Service links define the shape of a relationship with a partner by defining the messages and interfaces (WSDL port types) used in the interactions in both directions.

Associated with each business process instance is a state, this state is comprised of a set of containers that contain messages. Containers can be used as the destination of received message or invocation results or the source of a reply message or invocation parameters. The contents of a container can be also be accessed by certain basic activities, such as assign and switch.

The behaviour of a business process is specified using a set of activities. The execution of theses activities is structured using certain "structured activities": sequence, switch, pick, while, flow and scope.

- sequence activity: this activity contains one or more activities that are executed sequentially;
- switch activity: this activity contains an ordered list of one or more conditions and activity pairs. The conditions are considered in order and the first condition that evaluates to true has its associated activity executed. In the case where no condition holds true a default "otherwise activity" can be specified;

- while activity: this activity repeatedly executes an activity while its condition is true.
- pick activity: this activity contains an ordered list of one or more event and activity pairs. It awaits the occurrence of these events, and then executes the associated activity. Only one activity will be executed, even if multiple applicable events occur. The events which can be monitored are either “message events” or “alarm events”.
- flow activity: this activity provides concurrent execution the set of contained activities. These activities, and their sub-activities, can be “linked” to form “must occur before” relationships between activities.
- scope activity: this activity allows the contained activity to be associated with its own fault handlers and compensation handler.

To support communication with external Web services, three communication activities are provided: receive, reply, invoke.

- receive activity: this activity is used to wait for a particular message type from a particular partner, and place the contents of the message into a container. A receive activity can be flagged to create a process instance.
- reply activity: the activity is used to send a response to a request previously accepted through a receive activity, the contents of the message being obtained from a container.
- invoke activity: this activity can be used to invoke a service (either synchronous or asynchronous) provided by a partner. The request message will be obtained from a container, and if the invoke activity is synchronous, the response will be stored in a container.

Other activities include assign, wait and empty, and error handling activities: throw, terminate and compensate.

- assign activity: the activity can be used to assign (parts of) a message contained within one container to an other container.
- wait activity: the activity is used to introduce delays for a certain period of time or until a certain deadline is reached.
- empty activity: the activity can be used to introduce an activity that “does nothing.” This activity can be useful within compensation handlers that are not required to perform any changes.
- throw activity: the activity is used when a business process needs to signal an internal fault explicitly.
- terminate activity: the activity can be used to immediately abandon all execution within the current business process instance.
- compensate activity: the activity can be used to cause the initiation of compensation on a scope that has already completed its execution normally.

BPEL4WS does not assume that each business process instance or service instance has a distinct “endpoint,” or that interactions are based on a sophisticated transport infrastructure that can identify the involved participants, instead it provides support for “message correlation.” Message correlation enables message contents to be examined to

identify involved business process, even over multiple conversations. If interactions are based on a sophisticated message transport, message correlation may not be required.

2.4.2 Relevant Capabilities and Attributes

- Structured programming approach to process definition.
- Message correlation support.
- A centralized process coordination model.
- Compensation and exception handling constructs.

2.4.3 Evaluation Questions

- BPEL4WS process definition provides structured programming concepts, can ADAPT process definition be mapped to structured programming concepts?
- BPEL4WS support message collection over multiple conversations, how does ADAPT support coordination of such conversations?
- How does ADAPT support compensation and exception handling within processes?
- BPEL4WS only supports centralized process coordination, does ADAPT support decentralized process coordination?

2.5 Web Services Choreography Interface (WSCI)

2.5.1 Summary

The Web Service Choreography Interface (WSCI) [7] is an XML-based interface description language that describes the flow of messages exchanged by a web service participating in choreographed interactions with other services. The WSCI specification was authored by BEA Systems, Intalio, SAP AG and Sun Microsystems.

WSCI describes the observable behaviour of a Web service, but does not address the definition and the implementation of the internal process. WSCI describes behaviour in terms of temporal and logical dependencies among the exchanged messages, sequencing rules, correlations, exception handling, and transactions.

WSCI's interface description language, which is based on XML, is used to capture the modelling concepts of: interfaces, activities and choreographs of activities, processes and units of reuse, properties, context, message correlation, exceptions, transactions and compensation activities, and global model.

- Interfaces: The behaviour of a Web service is described as processes that are contained within the interface. A Web service may expose multiple interfaces for supporting multiple scenarios.
- Activities and choreographs of activities: The behaviour of the processes of interfaces is described in terms of choreographed activities. Activities can be atomic or complex. Atomic activities represent basic units of behaviour, such as sending and receiving messages. Complex activities are recursively composed from other activities. Complex activities support specific kinds of activity choreograph, such as: sequential execution, parallel execution, looping and conditional execution.

- Processes and units of reuse: The named units of behaviour in WSCI are processes, and are described by activities. Process can be reused, by referencing their names.
- Properties: References to values within the interface definition are modelled by properties. They are the equivalent of variables in imperative programming languages.
- Context: Scopes containing activities are modelled by contexts. They manage such thing as exception handling, and property definitions.
- Message correlation: To model the interrelationship between conversations WSCI has introduced message correlation. Different conversations can be distinguished by correlation instances, which are a set of properties' values.
- Exceptions: To model exceptional behaviour WSCI supports the definition of sets of activities that will be executed in response to particular exceptional behaviour.
- Transactions and compensation activities: WSCI contexts can be associated with a transaction. WSCI supports two models of transactional behaviour, atomic and open-nested. Atomic transactions have the standard ACID transaction properties. Open-nested transactions are composed of other transactions, which can be themselves atomic or open-nested transactions. The rollback of open-nested transactions is achieved by executing compensation activities.
- Global model: WSCI also allows modelling a multi-participant view of the overall message exchange by means of a global model. A global model is described by a collection of interfaces of the participant services, and a collection of links between the operations of the participant services.

2.5.2 Relevant Capabilities and Attributes

- Structured programming approach to process message exchange modelling.
- Compensation and exception modelling constructs.

2.5.3 Evaluation Questions

- Does ADAPT support the modelling of conversations?
- Does ADAPT support the modelling of compensation and exception handling in conversations?

2.6 Web Services Flow Language (WSFL)

2.6.1 Summary

The Web Services Flow Language (WSFL) [8] is an XML language for the description of Web services compositions. The WSFL 1.0 specification was published by IBM in May 2001, and has how been superseded by the BPEL4WS specification [6].

WSFL is intended to support two approaches to modelling Web service composition: Flow Models and Global Models. A Flow Model of composition is based on describing how to use the functionality provided by a collection of Web services. WSFL models these compositions by specified the flow of control and data between Web services. A Global Model of composition is based on describing how a collection of Web services interacts. The interactions are modelled as links between endpoints of the Web services'

interfaces, each link corresponding to the interaction of one Web service with an operation of another Web service's interface. Both Flow Models and Global Models support recursive composition, where Web service compositions can itself be provided as a Web service.

The WSFL specification describes the concepts of its models using a metamodel. The metamodel describes the entities that make up the models, and their interrelationships.

2.6.2 Relevant Capabilities and Attributes

- Flow model based composition.
- Global model based composition.
- Composition metamodel.

2.6.3 Evaluation Questions

- WSFL supports a flow model based composition, does ADAPT supports flow model based composition?
- WSFL supports a global model based composition, does ADAPT supports global model based composition?
- Would ADAPT benefit from having a composition metamodel?

2.7 Microsoft's XLANG

2.7.1 Summary

The purpose of Microsoft's XLANG specification [9] is to model the message exchange behaviour among Web services, and is expected to serve as the basis for an automated protocol engines that can track the state of process instances and help enforce protocol correctness in message flows. The XLANG specification was published by Microsoft in 2001 and is widely deployed as part of the BizTalk orchestration server, but has how been superseded by the BPEL4WS specification [6].

The goal of XLANG is to make it possible to formally specify business processes as stateful long-running interactions. The specific of such business processes are done in terms of the following:

- Sequential and parallel control flow constructs.
- Long running transactions with compensation.
- Custom correlation of messages.
- Flexible handling of internal and external exceptions.
- Modular behaviour description.
- Dynamic service referral.
- Multi-role contracts.

The XLANG approach to service description is to extend WSDL [11] service descriptions with extension elements that describe the behavioural aspects of the service. An example of such a behavioural description is:

```
<wsdl:service name="TransferService">
    . . .
    <xlang:behaviour xmlns:xlang="http://schemas.microsoft.com/biztalk/xlang/">
        <xlang:body>
            <xlang:sequence>
                <xlang:action operation="Debit" port="BankAccountA" activation="true"/>
                <xlang:action operation="Credit" port="BankAccountB"/>
            </xlang:sequence>
        </xlang:body>
    </xlang:behavior>
</wsdl:service>
```

2.7.2 Relevant Capabilities and Attributes

- Formally specify business processes.
- Support for stateful long-running interactions.
- Support for behavioural description of services?

2.7.3 Evaluation Questions

- Does ADAPT services descriptions contain behavioural information?

2.8 Web Services Conversation Language (WSCL)

2.8.1 Summary

The purpose of Web Service Conversation Language (WSCL) [10] is to provide a standard for specifying business level conversations. WSCL provides an XML schema for specifying business level conversations that take place at a single Web service. The WSCL specification was published as a W3C note by Hewlett-Packard in March 2002.

The WSCL notion of a *conversation* is a series of message exchanged between a service-consumer and a service-provider. The WSCL specification models a *conversation* as a finite state machine where state changes are triggered by *interactions*. An *interaction* is the exchange of one or two documents between a service-consumer and a service-provider. The WSCI model supports five types of interactions *Send*, *Receive*, *SendReceive*, *ReceiveSend* and *Empty* (the first four of which maps to the WSDL notions of: one-way, notification, send-response and requested-response).

2.8.2 Relevant Capabilities and Attributes

- Conversation model
- Interaction model

2.8.3 Evaluation Questions

- WSCL models conversations at service endpoints, does ADAPT require a conversation model?
- If ADAPT has a conversation model, does it have more powerful modelling capabilities than WSCL finite state machine?
- If ADAPT has an interaction model, does it have more powerful modelling capabilities than WSCL finite state machine?

- The WSCL conversation model only addresses conversations between two parties at a single endpoint, if ADAPT has a conversation model, does it address multi-party conversations?

2.9 Web Services Description Language (WSDL)

2.9.1 Summary

The purpose of the Web Services Description Language (WSDL) [11] is to address the need to describe network services. In WSDL, network service descriptions are contained in XML documents, which defines both the abstract and concrete entities required to specify network services in sufficient detail that they can be invoked, and related to other network service.

The entities, which are encoded as XML elements within a WSDL document, are:

- *types*: contain data type definitions using some type system, normally XML Schema.
- *messages*: are abstract definition of the content of messages being communicated.
- *port types*: are abstract definition of a set of operations supported by one or more endpoints.
- *bindings*: contains the specification of concrete protocol and data format to communicate.
- *services*: contain the specification of a collection of endpoints.

An illustration of the overall structure of a WSDL document, is given below:

```
<?xml version="1.0">

<wsdl:definitions name="..." targetNamespace="..." ...>
    <wsdl:types>...</wsdl:types> ?
    <wsdl:message name="...">...</wsdl:message> *
    <wsdl:portType name="...">...</wsdl:portType> *
    <wsdl:binding name="..." type="...">...</wsdl:binding> *
    <wsdl:service name="...">...</wsdl:service> *
</wsdl:definitions>
```

In the rest of this section the WSDL document elements, and their sub-elements, will be described in more detail.

The purpose of the *types* elements within a WSDL document is to allow the definitions of data types that can be included within *messages*. This is usually done using an XML Schema, though it should be noted that eventual wire format messages might not be in XML. The optional *types* element may be augmented by importing existing schemas into the WSDL document. An example of a WSDL document's types element is:

```
<wsdl:types>
    <xsd:scheme targetNamespace="http://example.com/job-control">
        <xsd:complexType name="NameValuePairType">
            <xsd:sequence>
                <xsd:element name="Name"  type="xsd:string"/>
                <xsd:element name="Value" type="xsd:string"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:scheme>
</wsdl:types>
```

Message elements are abstract definition of messages, which are exchanged between endpoints. Messages consist of zero or more logical *parts*, which are analogous to named and typed parameters. Each *part* is associated with a type from some type system, possibly defined in the *types* elements. An example of a WSDL document's message element is:

```
<wsdl:message>
    <wsdl:part name="AccountNumber" type="tns:AccountNumberType"/>
    <wsdl:part name="Amount"        type="xsd:integer"/>
</wsdl:message>
```

The purpose of the *port type* elements within a WSDL document is to allow the definition of abstract endpoints. The abstract endpoint definition is in terms of a set of *operation* elements. Each *operation* elements defines it name, and the message types involved in the operation. Depending on the messages involved in the operation four interaction models are supported:

- One-way: the endpoint receives a message.
- Request-response: the endpoint receives a message, and sends a correlated message.
- Solicit-response: the endpoint sends a message, and receives a correlated message.
- Notification: The endpoint sends a message.

Examples of port type elements, which are describing respectively: one-way, request-response, solicit-response and notification, are given below:

```
<wsdl:portType name="JobControl">
    <wsdl:operation name="Shutdown">
        <wsdl:input message="tns:ShutdownRequest"/>
    </wsdl:operation>

    <wsdl:operation name="GetStatus">
        <wsdl:input  message="tns:GetStatusRequest"/>
        <wsdl:output message="tns:GetStatusResponse"/>
        <wsdl:fault name="unknownService" message="tns:ErrorResponse"/>
    </wsdl:operation>

    <wsdl:operation name="AreYouAlive">
        <wsdl:output message="tns:AreYouAliveRequest"/>
        <wsdl:input message="tns:AreYouAliveResponse"/>
        <wsdl:fault name="unknownClient" message="tns:ErrorResponse"/>
    </wsdl:operation>

    <wsdl:operation name="Refresh">
        <wsdl:output message="tns:RefreshRequest"/>
    </wsdl:operation>
</wsdl:portType>
```

The purpose of the *bindings* elements are to define, for a particular *port type*, protocol details for operations, and the message format for associated messages. The *bindings* elements are extensible in that elements for other schemas can be introduced within the bindings. Specifications exist for bindings for SOAP, HTTP and MIME. Through these extensions the specifics of the protocol and message format can be specified for that particular from of binding. In the example below the SOAP binding extensions have been used:

```
<wsdl:binding name="JobControlSOAPBinding" type="tns:JobControl">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="GetStatus">
        <soap:operation soapAction=""/>
        <wsdl:input>
            <soap:body use="encoded" namespace="http://example.com/job-control"
```

```
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
        </wsdl:input>
        <wsdl:output>
            <soap:body use="encoded" namespace="http://example.com/job-control"
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
        </wsdl:output>
    </wsdl:operation>

    ...
</wsdl:binding>
```

In this example the protocol transport has been specified as HTTP, and that style of messages is RPC (which embeds messages within an element with the same name as the intended receiving method, as opposed to document which does not) and the messages will be encoded using SOAP encoding (which uses the SOAP schema as opposed to literal encoding where other arbitrary schemas are used).

The purpose of the *service* element is to define a set of network endpoints. Each network endpoint is defined by a *ports* element. In the example bellow, a service is specified containing a single network endpoint that is accessible at the specified URL, and is associated with a SOAP binding.

```
<wsdl:service name="JobControlService">
    <wsdl:port name="JobControl" binding="tns:JobControlSOAPBinding">
        <soap:address location="http://ws.acme.com/axis/services/JobControlService"/>
    </wsdl:port>
</wsdl:service>
```

2.9.2 Relevant Capabilities and Attributes

- Interaction model.
- Service description.
- External protocol specification.
- External QoS specification.
- Extensible.

2.9.3 Evaluation Questions

- Does ADAPT require a minimal quality of service for invocation?
- WSDL supports a service-oriented specification, does ADAPT support object-oriented specification of service?
- WSDL does not support inheritance in service specification, does ADAPT have support for inheritance in service specification?

2.10 Web Service Security (WS-Security)

2.10.1 Summary

The Web Service Security (WS-Security) [12] provides a framework to enable applications to construct secure SOAP message exchanges. The WS-Security 1.0 specification was published by IBM, Microsoft and VeriSign in April 2001.

The WS-Security specification describes enhancements to SOAP to allow message integrity and message confidentiality to be implemented. The WS-Security

enhancements are aimed to addressing two types of security threat: 1) messages being modified or read by antagonists or 2) an antagonist sending messages to a service that, while well-formed, lack appropriate security claims to warrant processing.

The WS-Security approach is to define the basic structure of a SOAP header for attaching security information to a SOAP message, and specifying of how encrypted information is encoded in a SOAP message. The types of security information that can be attached to SOAP message in a WS-Security SOAP header are:

- User name and password information (UsernameToken element)
- Binary security tokens, for example, X.509 certificates and Kerberos tickets (BinarySecurityToken element)
- References to security tokens what reside somewhere else and need to be "pulled" by the receiving application (SecurityTokenReference element)
- Key information, such as X.509 certificates (KeyInfo element)
- Message signature (Signature element)

2.10.2 Relevant Capabilities and Attributes

- Support for message integrity.
- Support for message confidentiality.

2.10.3 Evaluation Questions

- Does ADAPT require message integrity, and if it does how is it supported?
- Does ADAPT require message confidentiality, and if it does how is it supported?
- Does ADAPT require access control, and if it does how is it supported?

2.11 Universal Description, Discovery and Integration (UDDI)

2.11.1 Summary

The Universal Description, Discovery and Integration (UDDI) [13] project's purpose is to provide standardized methods of publishing and discovering information about Web services. The UDDI project is aiming to specify an open framework for describing service, discovering businesses, and integrate business services.

Conceptually, a business can register three types of information in a UDDI registry:

- Basic contact information and unique identifiers about the company.
- Categorisation of the Web services provided by the company.
- Behavioural description of the Web services provided by the company.

The UDDI Business Registry (UBR), also know as the "Public Cloud" has been deployed. The UBR is a decentralized registry in which the content is replicated over on all the nodes that operate the UBR.

The UDDI project has specified a number of XML schemas and programming APIs to allow publishing and discovery of information about business. The UDDI APIs are

based on five primary UDDI data structures types: businessEntity, publisherAssertion, businessService, bindingTemplate and tModel.

- *businessEntity*: contains the business's basic information, including contact information categorisation, description and identifiers.
- *publisherAssertion*: is used to indicate a relationships between two businesses (*businessEntity*s). This relationship is only made public if both businesses endorse the relationship.
- *businessService*: represent services provided by the business, both Web service and manual services. A *businessService* can be associated with one or more *businessEntity*s, and a *businessEntity* can contains one or more *businessService*s.
- *bindingTemplate*: contains pointers to the technical descriptions and the access endpoint of a service. A *businessService* can contain one or more *bindingTemplate*s.
- *tModel*: contains the abstract description of a particular specification or behaviour to which the service adheres. A *bindingTemplate* can contain one or more *tModel*s.

2.11.2 Relevant Capabilities and Attributes

- Support for publishing information about Web services.
- Support for discovering information about Web services.

2.11.3 Evaluation Questions

- Does ADAPT support mechanisms for publishing information about services?
- Does ADAPT support mechanisms for discovering information about services?

2.12 OMG Interface Definition Language (OMG IDL)

2.12.1 Summary

The OMG Interface Definition Language (OMG IDL) [14] is used to describe interfaces of CORBA objects. These interface definitions specify the operations the objects is equipped to execute, the input and output parameters required, and any exceptions that may be thrown. An interface can be derived from another interface, which is called a *base* interface of the *derived* interface. The *derived* interface supports all of the operations of the *base* interface in addition to its own operations. CORBA IDL allows interface definitions to be grouped into a module definition. An example of CORBA IDL is given below:

```
module CallbackTest
{
    interface PingPong;

    typedef sequence<PingPong> PingPongSeq;

    interface PingPong
    {
        boolean oper(in long level, in PingPongSeq objects);
    };
};
```

2.12.2 Relevant Capabilities and Attributes

- Service interface specification.

- Support of inheritance.

2.12.3 Evaluation Questions

- Does ADAPT support a human readable representation of a service specification?
- OMG IDL support inheritance in service specification, does ADAPT have support for inheritance in service specification?

2.13 CORBA Interface Repository (CORBA IR)

2.13.1 Summary

The CORBA Interface Repository (CORBA IR) [16] is the component of the ORB that provides persistent storage of interface definitions; it manages and provides access to a collection of object definitions specified in OMG IDL. Interface definitions are maintained in the Interface Repository as a set of objects that are accessible through a set of OMG IDL specified interface definitions. An interface definition contains a description of the operations it supports, including the types of the parameters, exceptions it may raise, and context information it may use.

2.13.2 Relevant Capabilities and Attributes

- Service specification API.

2.13.3 Evaluation Questions

- Does ADAPT support an API to allow service specification to be inspected?

3 References

[1] "Developing Enterprise Web Services", Sandeep Chatterjee and James Webber, Prentice Hall, 2003.

[2] Java 2 Platform, Enterprise Edition Specification (http://java.sun.com/j2ee)

[3] OASIS Business Transaction Protocol (BTP), Committee Specification 1.0 (https://www.oasis-open.org/committees/business-transactions/)

[4] Web Services Transactions (WS-Transaction) (http://www.ibm.com/developerworks/library/ws-transpec/)

[5] Web Services Coordination (WS-Coordination) (http://www.ibm.com/developerworks/library/ws-coor/).

[6] Business Process Execution Language for Web Service (BPEL4WS) (http://www.ibm.com/developerworks/library/ws-bpel/)

[7] Web Service Choreography Interface (WSCI) (http://www.w3.org/TR/wsci/)

[8] Web Service Flow Language (WSFL) (http://www.ibm.com/software/solutions/webservices/pdf/WSFL.pdf)

[9] Microsoft's XLANG (http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm)

[10] Web Service Conversation Language (WSCL) 1.0 (http://www.w3.org/TR/wscl10/)

[11] Web Services Description Language (WSDL) 1.1 Specification (http://www.w3.org/TR/wsdl)

[12] Web Services Security 1.0 (WS-Security) (http://www-106.ibm.com/developerworks/webservices/library/ws-secure/)

[13] Universal Description, Discovery and Integration (UDDI) 3.0 Specification (http://www.uddi.org/specification.html)

[14] OMG Interface Definition Language (OMG IDL) (http://www.omg.org/cgi-bin/doc?formal/02-11-03)

[15] CORBA Interface Repository (CORBA IR) (http://www.omg.org/cgi-bin/doc?formal/02-11-03)

[16] Microsoft .Net (http://www.microsoft.com/net/)

[17] CORBA Components (http://www.omg.org/cgi-bin/doc?formal/02-06-65)

[18] Java Management Extensions (JXM) (http://java.sun.com/products/JavaManagement/)

[19] The Physiology of the Grid (http://www.ggf.org/ogsi-wg/drafts/ogsa_draft2.9_2002-06-22.pdf)

[20] Grid Service Specification (http://www.ggf.org/meetings/ggf6/ggf6_wg_papers/draft-ggf-ogsi-gridservice-04_2002-10-04.pdf)