Transaction Management in Service-Oriented Systems: Implementation

Mr.Bhanawase V.V Mr. Mane S.M Mr.Joshi A.S

Abstract— New generation business models impose additional requirements on the IT support of business processes. In particular, quickly responding to new business requirements, continuously reducing IT cost, and dynamically integrating new business partners and customers are highly demanded. Recently, the trend in software development has shifted from developing software systems to developing service-oriented systems that are composed of ready to use services. The Service-Oriented Architecture (SOA) architectural style has been widely adopted in industry thanks to its ability of providing seamless integration among software services Service-based applications are constructed by integrating heterogeneous services that are developed using various programming languages and running on different operating systems from a range of service providers. Services are loosely coupled entities, often designed under open-world assumptions, distributed across organizational boundaries, and executed remotely at their service providers' environment. They require a smoother transition from development to operation than traditional applications. The objective of the paper is to look at the requirements of transaction management for Service oriented systems and the systematic requirements from the starting point for an analysis of current standards and technologies in the field of Web services

Key Terms— Transaction Management, Service Oriented Computing, BPEL

1. INTRODUCTION

Although the Web was initially intended for human use, most experts agree that it will have to evolve probably through the design and deployment of modular services to better support automated use. Services provide higher-level abstractions for organizing applications for large-scale, open environments. Thus, they help us implement and configure software applications in a manner that improves productivity and application quality. Because services are simply a means for building distributed applications, we cannot talk about them without talking about service-based applications specifically, how these applications are built and how services should function together within them. The applications will use services by composing or putting them together. An architecture for service based applications has three main parts: a provider, a consumer, and a registry. Providers publish or announce their services on registries, where consumers find and then invoke them. Standardized Web service technologies are enabling a new generation of software that relies on external services to accom- plish its tasks. The remote services are usually invoked in an asynchronous manner. Single remote operation invocation is not the revolution brought by Service-Oriented Computing (SOC), though. Rather it is the possibility of having programs that perform complex tasks coordinating and reusing many loosely coupled independent services.

A new approach to software, such as that brought by SOC, calls for new ways of engineering software and for new problems to be solved. The central role of these systems is played by services which are beyond a centralized control and whose functional and, possibly, non-functional properties are discovered at run-time. The key problems are related to the issue of discovering services and deciding how to coordinate them. For instance, while planning to drive to a remote city, one might discover that it is heavily snowing there, and may want to obtain snow tyers. Therefore, one needs to find a supplier and a transport service to have the appropriate tires in a specific location by a specific deadline. That is, various independent services are composed into the form of a process, called the 'get winter tyres while traveling' with the requirement that we order the tyres if and only if we find also a transport service for them. In other words, we require the services of tyre ordering and tyre delivery to be composed in a transactional manner. A service composition is a set of operations belonging to possibly many services, and a partial order relation defining the sequencing in which operations are to be invoked. Such a partial order is adequately represented as a direct graph. A service transaction is a unit of work comprehending two or more operations that need to be invoked according to a specific transaction policy.A service transaction can span over operations of one service or, more interestingly, of several services.

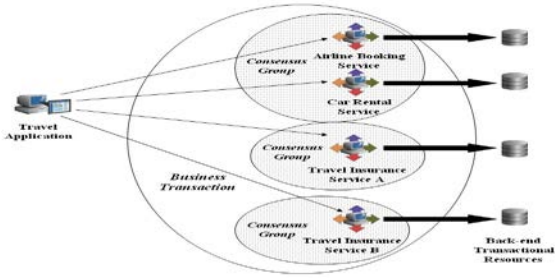

Fig.. Service-based travel application

WS-Coordination specification describes an extensive framework for providing various coordination protocols. The WS-Atomic Transaction and WS-Business Activity specifications are two typical Web transaction proto- cols. They leverage WS-Coordination by extending it to define specific coordination protocols for transaction processing. The former is developed for simple and short-lived Web transactions, while the latter for complex and long-lived business activities. Finally, the Business Process Execution Language (BPEL) is a process- based composition specification language. In order to develop reliable Web services compositions, one needs the integration of transaction standards with composi- tion language standards such as BPEL. Unfortunately, these are currently separate specifications.

This paper has a double goal: The first one is to look at the requirements of transaction management for Service-oriented systems. The systematization of requirements is the starting point for an analysis of current standards and technologies in the field of Web services. The second goal of the paper is to propose a framework for the integration of BPEL with transaction protocols such as WS-AtomicTransaction and WS-BusinessActivity.

The present work extends our survey and requirement analysis for service transactional systems and our proposal of the XSRL language for handling requests against service compositions. In XSRL a construct is defined to express atomicity of services execution, though no means for recovering from failures is provided.

The rest of the paper is organized as follows. First, we Introduce Service-based travel application example. We Continue by looking at web services and service oriented Architecture in Section 2 the proposed approach to Transaction management is presented in Section.

2. WEB SERVICES

Most service-based or business-to-business (B2B) applications would like to have the data consistency and correctness guarantees provided by existing transaction-processing platforms. Unfortunately, achieving these goals is difficult because interactions between the participants may be complex—involving multiple parties, spanning different organizations, and, most notably, lasting for hours or even days. For example, a transaction for a computer parts ordering application may involve different suppliers and may not be complete until all parts have been delivered. Should the various suppliers block part requests until this transaction is complete? Multiple reasons prevent such services from locking their back-end data resources for long durations: for example, the in ability to service additional incoming requests, and the potential for enabling denial-of-service attacks.

2.1 Web Services Transaction Architectures

Most web services transaction protocols build on the existing concepts of a transaction coordinator, participants, and a transaction context as shown in Figure 2.. The general concepts behind these protocols are analogous to those of traditional transaction systems.

As the figure indicates, the application interacts with the coordinator in some fashion (the specifics are unique to the transaction protocol utilized) to initialize a transaction and create an associated transaction context; the transaction context is propagated between the clients and the services, providing a flow of context information required to identify a transaction with which work should be associated (as well as the location of the coordinator). The propagation of the context may occur transparently to the client and services.

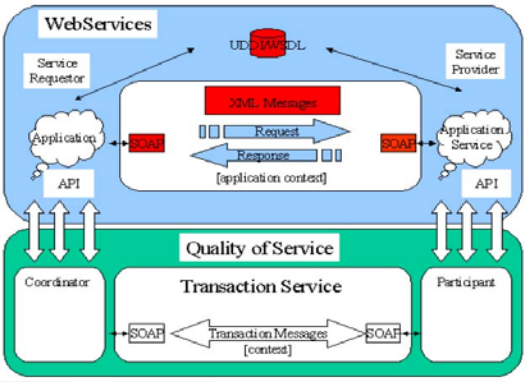

Fig . - Web services transactions

3. BUISNESS PROCESS EXECUTION LANGUAGE

The Business Process Execution Language (BPEL) for Web Services was proposed by BEA, IBM and Microsoft. The BPEL is a language for describing the behavior of business processes based on Web services. For the specification of a business process, BPEL provides activities and distinguishes between basic activities and structured activities. The basic activities include (receive) to provide web service operations and (invoke) to invoke web service operations. A structured activity defines a causal order on the basic activities and can be nested in another structured activity itself..

BPEL plays an important role in the web service standards family and is used in concert with a number of other standards. The Web Services Description Language (WSDL) is used to define the interface of a web service. BPEL relies on WSDL to ensure that the invocation of a service is correctly typed and also to deduce the synchronization behavior of this invocation. A web service is invoked by a BPEL interpreter using the Simple Object Access Protocol (SOAP), which defines the exchange of messages that are encoded in the eXtended Markup Language (XML) . The structure of these messages, as well as the syntax of BPEL itself is defined using XML Schemas.

4. TRANSACTION API

In a J2EE environment, the transaction manager has to communicate with the application server, the application program, and the resource managers, using a well-defined and standard API. The Java Transaction API (JTA) is defined

precisely for this purpose. The JTA defines a set of high-level interfaces that describe the contract between the transaction manager and three application components that interact with it: the application program, resource manager, and application server.

A. Java Transaction Service

The JTA specification's main purpose is to define how a client application, the application server, and the resource managers communicate with the transaction manager. Because JTA provides interfaces that map to X/Open standards, a JTA compliant transaction manager can control and coordinate a transaction that spans multiple resource managers (distributed transactions).

JTA does not specify how a transaction manager is to be implemented, nor does it address how multiple transaction managers communicate with each other to participate in the same transaction. That is, it does not specify how a transaction context can be propagated from one transaction manager to another. The Java Transaction Service (JTS) specification addresses these concepts.

JTS specifies the implementation contracts for Java transaction managers. It is the Java mapping of the CORBA Object Transaction Service (OTS) 1.1 specification. The OTS specification defines a standard mechanism for generating and propagating a transaction context between transaction managers, using the IIOP protocol. JTS uses the OTS interfaces (primarily org.omg.Cos Transactions and org.omg.Cos TSPortability) for interoperability and portability. Because JTS is based on OTS, it is not unusual to find implementations that support transactions propagating from non-Java CORBA clients as well.

A commonly used method in practice for supporting Web service transactions resorts to generic middleware. Some representative transaction management tools, such as IBM Web services Atomic Transaction for Web Sphere Application Server, JBoss Web service Transactions, and Apache Kandula, have employed.

This method to support distributed Web services atomic transactions. These tools focus on the implementation of transactions within some types of application servers using the Java Transaction API (JTA).

JTA provides three main interfaces, namely User Transaction interface, Transaction Manager Interface and Transaction interface. These interfaces share transaction operators, such as commit (), rollback (), suspend (), resume () and enlist (). The application servers act as Transaction Manager, and implement.

The coordination services described by the WS-Coordination specification .This method is practical and reuses the available generic middleware. Though, it does not take into account transaction management within BPEL processes. In Loechner presents a survey of issues in implementing transactions with service technology and proposes a model-based approach to managing transactions.

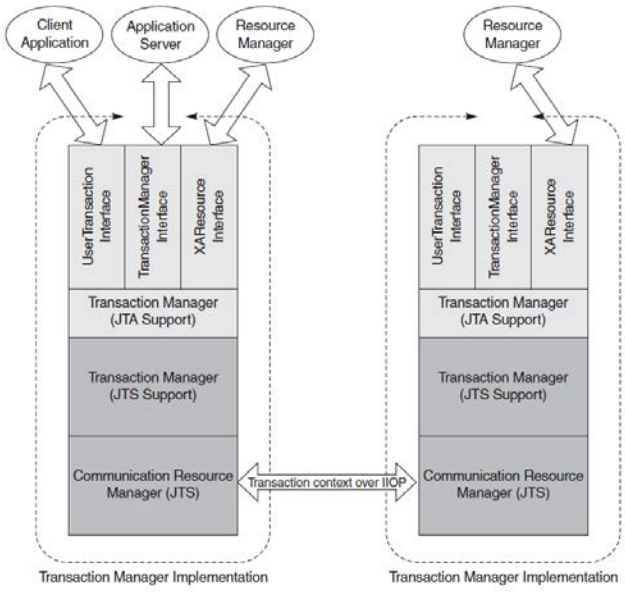

Fig - JTS transaction manager components

5. SOLUTION FOR TRANSACTION MANAGEMENT IN SERVICE ORIENTED SYSTEM

This part describes the System architecture and High level design of transaction management application. High level Design of the application shows how the whole of the process going on from the user request for an order to booking.

The system architecture in Figure-4 describes an enterprise level transaction between user, distributer and suppliers. The designed architecture based on distributed application processing in multiple tiers. All the distributed services are performs information exchange in support of web service which acts as a middleware. Enterprise level services are used to manage the transaction integrity service within user session.

5.1 SYSTEM MODULES

The designed system architecture in Figure-3.1 has the following modules:

1. User Interface
2. Transaction Manager
3. Distribution Service
4. Suppliers Service

1. User Interface: User interface module provides the user input screens. It takes the user requests and communicates with Transaction Manager for finding the availability of the requested order in the distributed suppliers. User interface provide inputs as Product name, delivery location and quantity of the product.

2.Transaction Manager: Transaction Manager on receive of request form user communicate with the distributed distribution service with support of web service. It implements Java Messaging service to communicate with the distribution service. On receive of reply from distribution

service it communicate with enterprise service to perform the integrity transaction in support java Transaction API and JDBC API for database transaction.

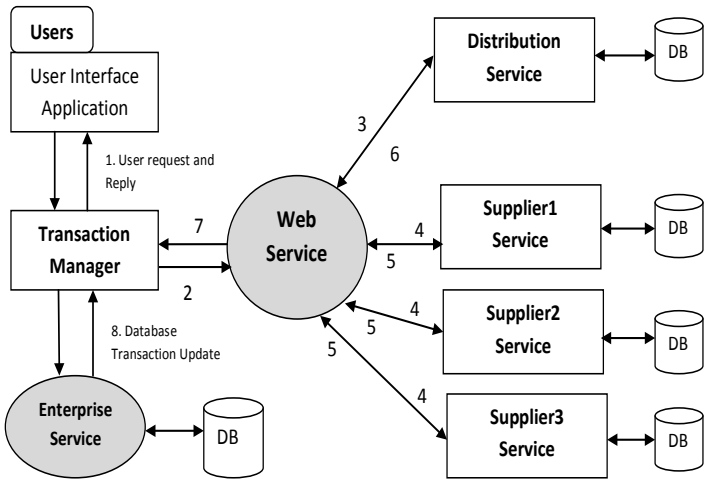

Fig .System Architecture

3.Distribution Service: Distribution service is a distributed service which maintains the suppliers' services information. On receiving the request from transaction manager it communicates with the registers suppliers with support of web service. It implements Java Messing Services for exchanging the information.

4.Suppliers Service: Suppliers service are also like distribution service which runs in distributed location. It maintains the product, location and carrier information for supply to the users. On receiving a request from distribution service it evaluates the available stock of the product and carriers for the delivery location requested.

6. IMPLEMENTATION

6.1 Technology overview

A. Java Technology

Java is a programming language originally developed by Sun Microsystems and released in 1995 as a core component of Sun's Java platform. The language derives much of its syntax from C and C++ but has a simpler object model and fewer low-level facilities. Java applications are typically compiled to byte code which can run on any Java virtual machine (JVM) regardless of computer architecture. Java is both a programming language and a platform. Java is fully object-oriented and everything in Java is an object. The goal of Java designers was to develop a language whereby the programmer could write the code once and could run this code anywhere, anytime forever.

B. JDBC

JDBC (Java Database Connectivity) provides a standard library for accessing relational databases. Using the JDBC API, you can access a wide variety of different SQL databases with exactly the same Java syntax. It is important to note that although JDBC standardizes the mechanism for connecting to databases, the syntax for sending queries and committing transactions, and the data structure representing the result, it does not attempt to standardize the SQL syntax. So, you can use any SQL extensions your database vendor supports. However, since most queries follow standard SQL syntax, using JDBC lets you change database hosts, ports, and even database vendors with minimal changes in your code. JDBC API is being used more and more in the middle tier of three-tier architecture. The JDBC API is also what allows access to a data source from a Java middle tier.

C. Java Messaging Service

The Java Message Service (JMS) API is a messaging standard that allows application components based on the Java 2 Platform, Enterprise Edition (J2EE) to create, send, receive, and read messages. It enables distributed communication that is loosely coupled, reliable, and asynchronous.

The JMS API supports two models:

- point-to-point or queuing model
- publish and subscribe model

In the point-to-point or queuing model, a producer posts messages to a particular queue and a consumer reads messages from the queue. Here, the producer knows the destination of the message and posts the message directly to the consumer's queue.

The publish/subscribe model supports publishing messages to a particular message topic. Zero or more subscribers may register interest in receiving messages on a particular message topic. In this model, neither the publisher nor the subscriber knows about each other.

D. Java Transaction API

The Java Transaction API (JTA), one of the Java Enterprise Edition (Java EE) APIs, enables distributed transactions to be done across multiple X/Open XA resources in a Java environment. The Java Transaction API consists of three elements: a high-level application transaction demarcation interface, a high-level transaction manager interface intended for an application server, and a standard Java mapping of the X/Open XA protocol intended for a transactional resource manager.

The **javax.Transaction.** User Transaction interface provides the application the ability to control transaction boundaries programmatically. This interface may be used by Java client programs or EJB beans.

The **UserTransaction.begin**() method starts a global transaction and associates the transaction with the calling thread. The transaction-to-thread association is managed transparently by the Transaction Manager.

Support for nested transactions is not required. The **UserTransaction.begin** method throws the **NotSupported Exception** when the calling thread is already associated with

a transaction and the transaction manager implementation does not support nested transactions.

6.2 Web Service Interface

To implement handle the message exchange we implement JMS Web services as middleware. An implementation of JMS interfaces for a Message Oriented Middleware (MOM) provided by JMS Providers which are implemented as either a Java JMS implementation or an adapter to a non-Java MOM. The elements of JMS Service are:

JMS client : An application or process that produces and/or receives messages.

- JMS producer : A JMS client that creates and sends messages.
- JMS consumer : A JMS client that receives messages.
- JMS message : An object that contains the data being transferred between JMS clients.
- JMS queue : A staging area that contains messages that have been sent and are waiting to be read. As the name queue suggests, the messages are delivered in the order sent. A message is removed from the queue once it has been read.
- JMS topic : A distribution mechanism for publishing messages that are delivered to multiple subscribers

6.3 Session Interface

To implement Session interface we need an EJB Application servers to support the UserTransaction interface and user sessionThe UserTransaction interface is exposed to EJB components through either the EJBContext interface using the getUserTransaction method, or directly via injection using the general @Resource annotation. Thus, an EJB application does not interface with the Transaction Manager directly for transaction demarcation; instead, the EJB bean relies on the EJB server to provide support for all of its transaction work as defined in the Enterprise JavaBeans Specification.

7.RESULTS

7.1 User interface screen

Fig 7.1– User Input Interface with user input as Product1, Location2 and Quantity 4

7.2 Middleware service

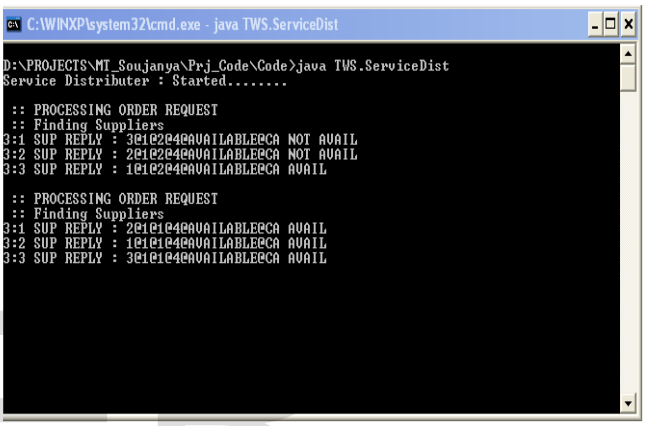

Fig 7.2 – Middleware Service – Service Distributors Service

The figure 7.2 will shows the interaction of the truncation manager with suppliers to evaluate the user input and there specified process requirements, it the supplier supplied items, location and item quality matches then the supplier values will identified as the available person, other wise it will indicate the supplier non available for the user to process the request.

7.3 Distributed Services Screen

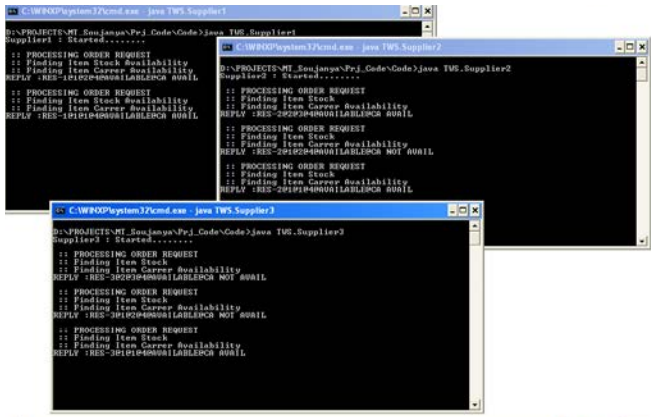

Fig 7.3 – Distributed Service – Supplier Service

The figures 7.3 will gives the description about the user interaction with the different suppliers on the similar time to process the user input request truncation evaluation proposes. The supplier process module will evaluate the user request and check the stock status and the quality of the item, if all requirements matches it will give the response as the available else the response is not available

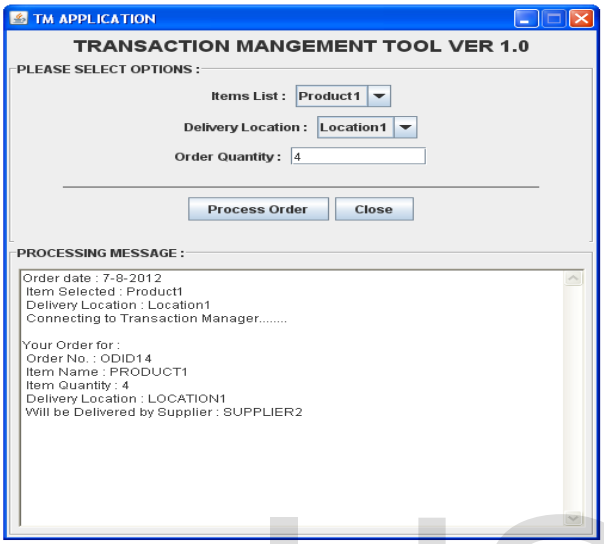

Fig 7.4 – User output Screen Interface

The figure 7.4 will show the user request process by which supplier and that location of the supplier and quantity of the items along the order will observe on the figure.

8. CONCLUSION

In this paper we explorer the key requirements of transaction management in Service-oriented systems and implement a order processing transaction management approach for Web service compositions. The key to implementing transaction management is to consider the combination of business logic with transactions, taking into account the challenges that make it impossible to directly apply transaction models to all processes.

REFERENCES

[1] WS-BA, "Web Services Business Activity Framework (WS Business Activity), Version 1.1," Arjuna Technologies Ltd, BEA Systems, Hitachi Ltd., IBM, IONA Technologies and Microsoft, Tech. Rep., 2007

[2] BPEL, "Business Process Execution Language for Web Services Version 1.1," IBM, Microsoft, BEAT, SAP and Siebel Systems, Tech. Rep., 2003.

[3] [Brown06] Brown W. And Cantor, M. SOA Governance: How to Oversee Successful Implementation through Proven Best Practices and Methods.IBM White Paper.
ftp://ftp.software.ibm.com/software/rational/web/whitepapers/10706900_SOA_gov_model_app_v1f.pdf

[4] A. Lazovik .M. Aiello, and M. Papazoglou, "Planning and monitoring the execution of Web service requests," International Journal on Digital Libraries, vol. 6, no. 3, pp. 235-246, 2006.

[5] M. Aiello and A. Lazovik, "Monitoring assertion-based business process," International Journal of Cooperative Information Systems, vol. 15, no. 3, pp. 359-390, 2006.

[6] B. Haugen and T. Fletcher, "Multi-party electronic business transactions. Version 1.1," UN, Tech. Rep., 2002.

[7] M.Little, "Transactions and web services," Communication of the ACM, vol. 46, no. 10, pp. 49-54, 2003.

[8] OASIS, "Business transaction protocol," OASIS, Tech.Rep., 2004.

[9] BPEL, "Business Process Execution Language for Web Services Version 1.1," IBM, Microsoft, BEAT, SAP and Siebel Systems, Tech. Rep., 2003.

[10] C. Sun, D. Hammer, G. Biemolt, and H. Groefsema, "An evaluation of desctiption- and management- standards and languages for Web service transactions," Univ. of Groningen/ SeCSE Project, Tech. Rep., 2006.

[11] [Gold-Bernstein05] Gold-Bernstein, B. and So, G.Integration and SOA: Concepts, Technologies and best Practices.

[12] [High05] High, R., Kinder, S., and Graham, S. IBM's SOA Foundation: An Architectural Introduction and Overview. November 2005. http://download.boulder.ibm.com/ ibmdl/pub/ software/dw/webservices/wssoawhitepaper.pdf [IBM06]

[13] B. Haugenand T. Fletcher, "Multi party electronic business transactions.Version1.1"UN Tech.Rep.,2002.

[14]M. Little, "Transactions and web services" Communication of the ACM,vol.46,no.10,pp.49-54,2003.

[15] C.Ouyang,E.Verbeek ,W.M. vander Aalst ,S. Breutel , M.Dumas, andA.Ter Hofstede, "Formal semantics and analysis of control in BPEL" Sci. Computer. Program, vol.67
pp. 162-198,2007.

[16] G.Chiola,"Area chability graph construction algorithm based on canonical transition ring count vectors ," in Petri Net sand Performance Models, 2001,pp.113-122.

[17] S. Consortium, "http://www.secse-project.eu/," European Union, Tech. Rep., 2005-2007.

[18] The SECSE Team, "Designing and deploying service-centric systems: The se cse way." in Service Oriented Computing: a look at the Inside (SOC@Inside'07), 2007.

[19] Various Authors, "Report on methodological approach to designing service compositions (final), version 4.0 Se CSE A3.D3," ESI, CA and CE-FRIEL, Tech. Rep., 2005, http://www.secseproject.eu/.

[20] "Report on methodological approach to design service compositions (v2.0) Se CSE A3.D3.2.b," CEFRIEL Uni sannio, Tech. Rep., 2006, http://www.secse-project.eu/.

[21] Active BPEL, "Active bpel engine 2.0," 2009, http://www.activebpel.org.