# Requirements and Evaluation of Protocols and Tools for Transaction Management in Service Centric Systems

Changai Sun, Marco Aiello
*Department of Computing Science, The University of Groningen,*
*Blauwborgje 3, 9747 AC, Groningen, The Netherlands*
*{csun, aiellom}@cs.rug.nl*

## Abstract

*As Service Centric (SC) Systems are being increasingly adopted, new challenges and possibilities emerge. Business processes are now able to execute seamlessly across organizations and to coordinate the interaction of loosely coupled services. Often it is necessary to have transactionality for a set of business operations, but the loosely nature of such systems calls for techniques and principles that go beyond traditional ACID transactions. By analyzing existing service composition languages, tools, and needs on a classical example, we provide requirements for transactionality in Service Centric Systems and indications for developing SC systems transactionally capable.*

## 1. Introduction

The widespread adoption of Web services is feeding the promises of the new field of Service Centric Systems. In a SC system, a computation is carried out by asynchronous messaging among independent programs which publish their functionalities in a standard way and are available over a network. In the case of *Web services* the format of the messages, of the description of the functionalities and interactions is based on XML protocols [13]. Then, one can compose services to form business processes discovering them, for instance, on the Internet. In this way, Web services become a mean to integrate operations and applications at the inter-enterprise level [11].

Consider for instance the case of a supply-chain spanning across independent organizations. By using Web services it is possible to allow the interaction of the independent information systems with relative ease, but the nature of SC systems implies the possibility of unforeseeable failures and the impossibility of centralizing the control. If, for instance, one is assembling a travel package, it makes little sense to pay for a hotel in Barcelona if no plane is available to get to the desired location. In other words, some operations depend on the successful completion of other operations and should be conducted in a transactional way.

If transactions have been extensively studied in the context of databases, where usually one desires to have ACIDity (Atomic, Consistent, Isolated, Durable transactions), in SC systems the desiderata are different. In fact, having lower control over the operations, their execution and outcomes, different properties are possible and desirable.

In this paper, we identify requirements for transaction management in SC systems, and evaluate the satisfaction of them by existing Web service transaction standards, service composition languages, and transaction management tools. The analysis does not only offer a survey but it is useful for developing new standards, techniques and tools for transaction management and boost the possibilities offered by SC systems.

The rest of the paper is organized as follows: in Section 2 we describe the Drop Dead Order (DDO) example which we use throughout the paper. In Section 3, we consider the set of transaction requirements for SC systems. The evaluation of services composition languages and tools with respect to these transaction requirements is presented in Sections 4 and 5, respectively. Discussion of the evaluation, conclusions and open issues conclude the paper (Section 6).

## 2. The Drop Dead Order Example

The drop-dead order describes a scenario where a customer wants to order products from a distributor under the condition that the products are delivered before the drop-dead date (Figure 1). The example is inspired by the example described by Bob Haugen and Tony Fletcher [10] and extensively refined by other authors, e.g., [16].

In the scenario, the distributor tries to find a supplier that has the products available. If he finds such a supplier, he will search for a carrier that is able to deliver the products before the drop-dead date. If both the supplier and the carrier are able to fulfill the demands of the customer, the distributor reports to the customer that he can fulfill the order. After the customer has acknowledged, the distributor sends a confirmation to the supplier and the carrier.

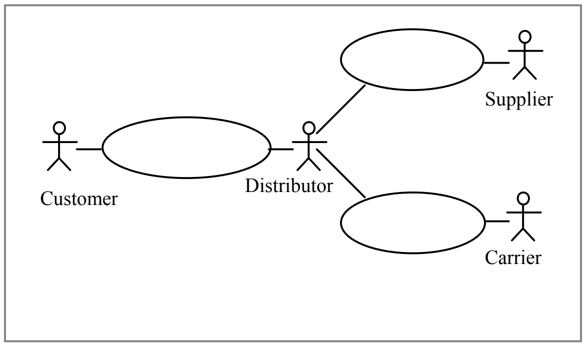

**Figure 1. The Drop Dead Order Example**

## 3. Transaction Requirements

In the field of databases, transactions are required to satisfy the so called ACID properties, that is, the set of operations involved in a transaction should occur atomically, should be consistent, should be isolated from other operations, and their effects should be durable in time. Given the nature of SC systems, satisfying these properties may result impossible and, in the end, not necessarily desirable [11]. In fact, some features are unique of SC systems:

• *Long-lived and concurrent,* unlike traditional transactions where they are usually short and sequential.

• Distributed over *heterogeneous* environments.

• Greater range of *transaction types* due to different types of business processes, service types, information types, or product flows.

• Unpredictable *number of participants*.

• Unpredictable *execution length.* E.g., information query and flight payment needs 5 minutes; while e-shopping an hour; and a complex business transaction like contracting may take days.

• *Greater dynamicity.* Computation and communication resources may change at runtime.

• Greater *security and privacy* concerns. More stringent requirements on authentication, information encryption and non-repudiation.

• Unavailability of *undo operations*, most often only compensating actions that return the system to a state that is close to the initial state are available.

These emerging features are distinctive of SC systems and need to be addressed in new ways. Blocking protocols, such as the two phase commit, guarantee atomicity and consistency, but are not directly usable in SC systems for the following reasons:

• Blocking may result in deadlocks especially when the number of concurrent transactions is high.

• ACID transactions may be difficult to implement due to the change in participants and the heterogeneous environments (especially for long-lived transactions).

• Often, it is not feasible to control dynamic resources located in another administrative domain.

Given these observations, one is left to wonder what are then the requirements for transactions in SC systems. We answer this next by grouping the requirements with respect to ACID properties and adding a fifth set of properties which goes beyond ACIDity. The drop dead order example is used for illustrating them.

*1.0 Atomicity* is the property of a transaction to either succeed successfully or not at all, even in the event of partial failures. In the DDO example, it should not happen that the supplier's resources are committed while the Carrier is not.

*1.1 Rollback* is the operation of returning to a previous state in case of a failure during a transaction. This may be necessary to enforce consistency. In the DDO, when the Distributor assigns a Supplier but cannot assign a Carrier, the changes made with the Supplier (and Customer) should be rolled back.

*1.2 Compensating actions* are executed in the event of a failure during a transaction, all changes performed before the failure should be undone. If the Distributor assigned a Supplier and committed it but cannot assign a Carrier, the changes made with the Supplier (and Customer) should be compensated.

*2.0 Consistency* is the property of a transaction to begin and end in a state which is consistent with the intended semantics of the system, i.e., not breaking any integrity constraints. A state in which the Carrier is committed but has never prepared to commit is inconsistent.

*2.1 Abort* is the returning to the initial state in case of failure or if the user wishes so. When the Distributor assigns a Supplier but cannot assign a Carrier, the entire transaction is to abort.

*2.2 Adding deadlines to transactions* involves giving timeouts to operations. Suppose that the Customer needs the goods before a certain time, then the Distributor and the Carrier need to comply with certain time constraints, too.

*2.3 Logical expressions for specifying constraints* are used for giving unambiguous and semantically defined rules for guaranteeing consistency. For instance, the fact that the account of the Distributor cannot be debited while the account of the Customer is not credited in the event of a money exchange can be expressed by *debited(distributor) + credited(customer) = 0*.

*3.0 Isolation* is the property of a transaction to perform operations isolated from all other operations. One transaction can therefore not see the other transaction's data in an intermediate state. The Customer should not be aware of the state of the transaction between the Distributor and the Supplier/Carrier regarding a different order.

*4.0 Durability* is the property of a transaction to record the effects in a persistent way. Whenever a transaction notifies one participant of successful completion, the effects must persist, even when subsequent failures occur. When the Supplier is notified of a successful completion, but somehow the connection with the Carrier fails, the changes with the Carrier should still be made.

*5.1 Composite transactions* are nested transactions. In the DDO example, the distribution transaction consists of two sub-transactions, namely, the supply and the deliver transactions. These transactions depend on the global outcome, that is, all three succeed or the whole composite transaction fails.

*5.2 Distributed transactions* are transactions between two or more parties executing on different hosts. The transaction should support transactions through a network between two different hosts. A customer can place a drop-dead order at the Distributor through a network connection.

*5.3 Transaction recovery by dynamic rebinding and dynamic re-composition at runtime* is the possibility of forming a new binding at runtime with a different party when the current service is not able to fulfill its promises. Dynamic re-composition is the forming of a new composition by replacing one or several services by another composition that fulfils the same function. Imagine that the first Carrier somehow fails and is unreachable. If this happens during a transaction, then automatic re-bind with a service that offers the same service should take place. Re-composition through re-binding with a third Carrier through the Supplier is also a possibility.

*5.4 Secure transactions of different types (Confidentiality, Integrity, Authentication and Non-repudiation)* refer to the fact that participants in a transaction may be authorized and authenticated. Data integrity should always be maintained. Also, mutual agreements cannot be denied after engaging in the transaction. To support a secure distribution transaction, such as in the DDO example, an authentication, authorization or encryption protocol should be supported by the transaction mechanism.

*5.5 Optimistic or pessimistic concurrency control* refers to the support of different types of concurrency control to enforce consistency. This control could either be optimistic or pessimistic. The pessimistic approach prevents an entity in application memory by locking it in the transaction for the entire time. While the optimistic simply chooses to detect collisions and then resolve the collision when it does occur. This scheme has better performance. When two transactions are concurrent, they should not both claim the same supply of goods from one Supplier.

For the Drop Dead Order example, we see that all these requirements are necessary with the exception of *5.4* and *5.5*. Existing transaction protocols are based on pessimistic concurrency control (locking). But let us look at this in more detail by considering, first existing standards and composition languages, and then tools referring to the just listed requirements.

## 4. Transaction Standards and Service Composition Languages

WS-Transactions [3,4] and Business Transaction Protocol (BTP) [14] are the two most representative standards that directly address the transaction management of Web service-based systems, while for representing compositions of services the Business Process Execution Language (BPEL) [7] and the Choreography Description Language (WS-CDL) [18] are most widely known and adopted.

WS-Transactions consists of two coordination protocols: WS-AtomicTransaction (WS-AT) [3] and BusinessActivity (WS-BA) [4] which live in the WS-coordination framework [5]. WS-AT provides the coordination protocols for short-lived simple operations, while WS-BA provides the coordination protocols for long-lived complex business activities. The WS-coordination framework is extensible and incremental. That is, WS-coordination can enhance existing SC systems with transaction properties by wrapping them with a specific coordination.

On the other hand, BTP [14] is a model for long-lived business transaction structured into small atomic transactions, and using cohesion to connect these atomic operations. Its motivation is to optimize the use of resource involved in a long-lived transaction under loosely coupled Web service environments and avoiding the use of a central coordinator.

BPEL [7] provides the facilities to specify executable business processes with references to services' interfaces and implementations. It does handle some basic issues of transactions, such as compensation, fault and exception handling, but other transaction requirements are not managed.

WS-CDL [18] provides the infrastructure to describe cross-enterprise collaborations of Web services in a choreographic way. The transactions are not explicitly addressed, but some facility can be used to satisfy some basic transaction properties, as we see next.

Let us now consider the proposed protocols that take the transaction and the business perspective of SC systems with respect to the requirements identified in Section 3 (for further details we refer to [16]). In Table 1 we summarize the results of the evaluation for all requirements—each row—and for all protocols—each column—by denoting the satisfaction with the 'y' symbol, the partial satisfaction with 'p', and no support with 'n'.

**Table 1. Evaluation Results**

| Reqs | BTP | WS-AT | WS-BA | BPEL | WS-CDL |
|---|---|---|---|---|---|
| 1.0 | y | y | n | p | p |
| 1.1 | y | y | p | p | y |
| 1.2 | n | n | y | y | p |
| 2.0 | y | y | p | p | p |
| 2.1 | y | y | y | y | n |
| 2.2 | n | y | y | p | y |
| 2.3 | n | n | n | y | y |
| 3.0 | n | y | y | y | y |
| 4.0 | y | y | y | y | p |
| 5.1 | y | y | y | y | y |
| 5.2 | y | y | y | y | y |
| 5.3 | n | n | n | y | n |
| 5.4 | n | n | n | p | p |
| 5.5 | n | n | n | n | y |

First, we remark that WS-Transaction actually consists of two different protocols with different properties, which we analyze separately. WS-AT is a traditional protocol which satisfies the basic ACID properties. WS-BA, on the other hand, renounces to atomicity to accommodate long-lived transactions. BTP has included confirmsets. These confirmsets let the application element choose which operations with parties in the transaction are to be cancelled and which are to be confirmed. In this way, the application element is able to contact more services which perform the same task and to choose the best option. Unfortunately, BTP is not part of the WS-Stack, which limits its compatibility with other Web service technologies. In addition, BTP does not support long-lived transactions. There is also a difference in granularity between the above transaction standards. WS-AT contains simple two phase commit protocols, WS-BA contains non-blocking protocols and BTP consists of a sequence of small atomic transactions.

As for security, WS-Security [15] can be combined with WS-Transaction as well as with BTP.

Dynamic rebinding is supported only by BPEL, though only at the implementation level. WS-CDL supports most requirements, while its major disadvantage is that the large players in the field do not support it and that no implementation is available.

We can further draw the following conclusions in terms of extensions to the traditional transaction model. WS-AT is a very conservative business transaction model especially with respect to blocking. WS-BA is more appropriate for services, by renouncing to the concept of the two-phase commit. BTP places itself in the middle (two phase commit is followed in a relaxed way). As for BPEL and WS-CDL they address the business process perspective with limited transaction support.

## 5. Transaction Management Tools

As the standards and protocols for Web services become more popular and stable, the number of tools supporting them increases. When such a tool is integrated into a service composition platform, usually one requires that it is complete, standalone (to be easily integrated), and open source. It should also have sufficient documentation and maintenance.

We propose a framework for comparing transaction management tools, which consists of a list of *concerns* indicating important aspects one needs to consider when integrating or reusing the tool. The tools evaluated include commercial products and open source software. Some tools are standalone transaction managers while others are integrated into application server containers. In Table 2 we summarize the evaluation results (see [16] for more details). The rows of the table consist of the following distinctive features.

*Functionality*: which transaction protocols are supported, such as WS- AT, WS- BA, and BTP.

*Status:* which indicates the maturity of the tool.

*Platform:* support at the level of Operating System, Programming Language, Container, etc.

*Documentation*: the availability and quality of installation instructions, tool architecture design descriptions, and developer guidelines.

*Integrability*: the degree to which the tool can be easily integrated and how well are the APIs described.

*Support & Maintenance*: the degree of industrial or community support for the tool.

*Cost:* under which license is the tool released.

*Others factors:* other features which describe the qualities of the tool include extensibility, reliability, usability, scalability, performance.

The columns of Table 2 represent the most adopted and best-known tools available today and are: *Apache Kandula* [2] whose aim is to provide an open-source implementation of WS-Coordination, WS-AT and WS-BA based on Apache Axis. *IBM WS-AT for WAS* [8] provides transactional support for Web service application requests that are compliant with Java Specification Requests and made using SOAP/HTTP. It supports WS-AT based on the WebSphere Application Server. *JBoss Transactions* [9] is a solid platform for distributed transactions. It has full support for WS-Coordination, WS-AT and WS-BA. It can be integrated into the JBoss Application Server or used as a stand-alone transaction manager for Java applications. *OpenWS-Transaction* [12] implements the WS-Transaction standards. It is part of a thesis project at the University of Georgia. *Choreography's Cohesions* [6] is a Business Transaction Management (BTM) tool suite enabling the management and coordination of loosely coupled applications in heterogeneous environment. *ActiveBPEL Engine 2.0* [1] is an Open Source implementation of a BPEL engine.

## 6. Discussions and Open Issues

Given the requirements for transactions in service centric systems and the evaluation of protocols and tools, we suggest that WS-Transaction should be selected as a model, BPEL as a service composition language, and JBoss Transactions as transaction management tool. The most notable reasons for these choices are the following ones. WS-Transaction is preferred since it supports long-lived transactions and is part of the WS-Stack. BPEL is preferred because of its industrial support and wide adoption. JBoss Transactions is preferred because it is a complete, standalone, open source tool, it has sufficient documentation. Currently, BPEL and WS-Transaction are independent specifications which strongly need integration. However, it is still not known how to integrate the transaction management into the service compositions.

In [17] we "follow what we preach" and provide a design, architecture and implementation of transaction management into a service centric platform according to the recommendations and evaluations provided here. By the results presented in this paper and in [17], we remark that the alignment between the business and the transaction perspective in SC systems still needs to be reconciled, but that there is space for integration. Furthermore, we have been able to identify features for evaluation and open issues which need further investigation, such as optimistic concurrence controls during transactions, introduction of transaction policies, and adaptation of transaction management tools in the context of Web services.

## Acknowledgements

We thank Dieter Hammer, Gerard Biemolt and Michiel Koning for fruitful discussion and support in the presented evaluation. This work is supported by the EU Integrated Project SeCSE (IST Contract No. 511680): **S**ervice **C**entric **S**oftware **E**ngineering.

## References

[1] ActiveBPEL, ActiveBPEL Engine 2.0, http://www.activebpel.org/?googleoss
[2] Apache, Kandula, http://ws.apache.org/kandula/
[3] Arjuna Tech. Ltd., BEA Systems, Hitachi Ltd., IBM, IONA Tech. and Microsoft, Web Services Atomic Transaction (WS-AtomicTransaction), v 1.0, August 2005
[4] Arjuna Tech. Ltd., BEA Systems, Hitachi Ltd., IBM, IONA Tech. and Microsoft, Web Services Business Activity Framework (WS-BusinessActivity), v 1.0, August 2005
[5] Arjuna Tech. Ltd., BEA Systems, Hitachi Ltd., IBM, IONA Tech. and Microsoft, Web Services Coordination (WS-Coordination), August 2005
[6] Choreology, Cohesions, http://www.choreology.com/
[7] IBM, Microsoft, BEAT, SAP and Siebel Systems, Business Process Execution Language for Web Services v 1.1, 2003
[8] IBM, Web Services Atomic Transaction for WebSphere Application Server (WS-AT for WAS), http://www.alphaworks.ibm.com/tech/wsat
[9] JBoss, JBoss Transactions, http://labs.jboss.com/portal/jbosstm
[10] B. Haugen, T. Flectcher, Multi-Party Electronic Business Transactions, Version 1.1, December 2002
[11] M. Little, Transactions and Web Services, CACM, Vol. 46, No. 10, 2003, pp 49-54
[12] I.Vasquez, OpenWS-Transaction, http://lsdis.cs.uga.edu/~vasquez/index.php?page=2,
[13] M.P. Papazoglou, Web Services Technologies and Standards, submitted to ACM Computing Surveys, 2007
[14] OASIS, Business Transaction Protocol, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=business-transaction, 2004
[15] OASIS, WS-Security Core Specification 1.1, http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf, 2006
[16] C. Sun, D. Hammer, State of the Art on Transaction Management in Service Centric Systems, Technical Report, Univ. of Groningen, Aug. 2006
[17] C. Sun, M. Aiello, Transaction Management for SeCSE, Technical Report, Univ. of Groningen, to appear.
[18] W3C, Web Services Choreography Description Language, v. 1.0, 2005

Table 2. Comparison of Transaction Management Tools

| | Apache Kandula | WS-AT for WAS | JBoss Transactions | OpenWS-Transaction | Cohesions | Active Engine 2.0 |
|---|---|---|---|---|---|---|
| **Functionality** | Support WS-AT & WS-BA. | Support WS-AT. | Support WS-AT, WS- BA and WS-C. | Support WS-AT, WS- BA and WS-C. | Support BTP, WS-C, partially WS-AT and WS-BA. | No support. |
| **Status** | WS-AT &WS-C ready; WS-BA not ready. | Ready, published by IBM in September 2003. | Ready, published by JBoss. | Only the prototype published. | Complete Product published. | Ready, Published by ActiveBPEL, February 2006. |
| **Platform** | Support Windows; Written in Java; Used with Apache Axis and Apache Tomcat. | Support Windows 2000 & Windows XP; Written in Java; Used with WebSphere Application Server, J2EE, JTA. | Support Windows 2000 Professional/Server, Windows XP Professional Sun Solaris 8 (Sparc), HP-UX and Redhat Linux 7.3; Written in Java; Used with webMethods Glue 5, Apache Axis on JBoss , WebLogic, SQL Server 2000 or Oracle 8, 9,10. | Support Windows and Unix; Written in Java; Used with Apache Axis, Apache Tomcat, BerkeleyDB, ActiveBPEL. | Support Windows 2000/XP/2003, Solaris 9 and 10, Red Hat Linux AS/ES 3.0. Written in Java; Used with 2RE1.4.2_02 and Most application server containers. | Support Windows and Unix; Written in Java; Used with Tomcat 5.5 and Java 1.5. |
| **Documentation** | Architectural Design and User Guide. | Installation instruction for Windows. | Installation guide, Programmer's guide, Administration guide, Javadoc. | Little documentation and papers available from author's website. | Administrator's Guide Programmer's Guide, Integration Guides for each application container, JTA and Jini Integration Guide. | Architecture, Developer's guide, User's guide, Engine installation, BPEL deployment, Tutorial, Samples. |
| **Extensibility and Integration** | Interoperable with other implementations , particularly those by Microsoft and IBM. | Integrated with Websphere application server and Interoperate with Microsoft's .NET environment. | Supports pluggable Web Service transaction protocols; It can be used as stand-alone transaction manager or be integrated with JBoss Application Server. | Not explicitly addressed. | Support for many application server containers, such as Apache Axis and BEA Weblogic, Websphere, and so on. | Easy to be integrated or extended. |
| **Support and Maintenance** | No special support and two versions are released. | Supported by IBM. | Professional support, consulting and training are available from JBoss. | No support or maintenance available. | Support available from Choreology, Continuous releases (1.0, 2.0, and 3.0). | Strong support from Active Endpoints Inc. Continuous releases (1.0, 2.0 and 3.0). |
| **Cost** | Open Source Software released under GPL. | Copyrighted and licensed by IBM, Commercial license is charged. | Open source software Released under GPL. | Open source, free download. | Commercial Product. | Open source software released under GPL. |