R.Soujanya, Guide: K.Venkateshwarlu / International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622 www.ijera.com
Vol. 2, Issue 5, September- October 2012, pp.1914-1921

The Transaction Concept: Integrated With BPEL and WS-AT, WS-BA Coordination Protocols

R.Soujanya (M.Tech) **Guide: M.Venkateshwarlu** M.Tech
Department of SE, Sri Kottam Tulasi Reddy Memorial College, Kondair. Andhra Pradesh.
Department of CSE, Associate Professor CSE Department, Sri Kottam Tulasi Reddy Memorial College, Kondair. Andhra Pradesh.

ABSTRACT

Service-Oriented Computing (SOC) is becoming the mainstream development paradigm of applications over the Internet, taking advantage of remote independent functionalities. When the control over the communication and the elements of the information system is low, developing solid systems is challenging. In particular, developing reliable web service compositions usually requires the integration of both composition languages, such as the Business Process Execution Language (BPEL), and of coordination protocols, such as WS-Atomic Transaction and WS-Business Activity. Unfortunately, the composition and coordination of web services currently have separate languages and specifications.

The goal of this paper is twofold. First, we identify the major requirements of transaction management in Service-oriented systems and survey the relevant standards. Second, we propose a semiautomatic approach to integrate BPEL specifications and web service coordination protocols, that is, implementing transaction management within service composition processes, and thus overcoming the limitations of current technologies.

Keywords: Web services, transaction management, WS-Business Activity, WS-Atomic Transaction, ACID properties, Service composition languages, and Business process execution language.

1. INTRODUCTION

The widespread adoption of Web services is feeding the promises of the new field of Service Centric Systems. As Service Centric (SC) Systems are being increasingly adopted, new challenges and possibilities emerge. Standardized web service technologies are enabling a new generation of software that relies on external services to accomplish its tasks. The remote services are usually invoked in an asynchronous manner. They are known by their published interfaces, and await invocations over a possibly open network. 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.

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.

In the present treatment, a service is a standard XML description of an autonomous software entity, it executes in a standalone container, it may have one or more active instantiations, and it is made of possibly many operations that are invoked asynchronously. 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. The coordination of a service transaction is the management of the transaction according to a given policy. A service transaction can span over operations of one service or, more interestingly, of several services.

One may argue that transaction management is a well-known Technique for the new features of transactions executed by web services; various web transaction specifications have been developed. WS-Coordination [1] specification describes an extensive framework for providing various coordination protocols. The WS-AtomicTransaction (WS-AT) [2] and WS-Business Activity (WS-BA) specifications [3] are two typical web transaction protocols. 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) [4] is a process-based composition specification language. In order to develop reliable web services compositions, one needs the integration of transaction standards with composition language standards such as BPEL [5], [6]. Unfortunately, these are currently separate specifications.

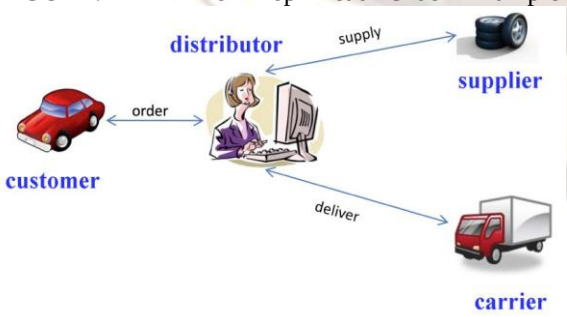
This paper has a double goal: the first one is to look at the requirements of transaction management for Service-oriented systems. The systematization of requirements is the starting point for an analysis of current standards and technologies in the field of web services. The second goal of the paper is to propose a framework for the integration of BPEL with transaction protocols such as WS-AtomicTransaction and WS-Business Activity. We use a simple but representative example across the paper, the drop-dead order (DDO) one, to illustrate requirements and the proposed approach.

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).

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: 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 Service-Oriented Systems, satisfying these properties is often not possible and, in the end, not necessarily desirable. In fact, some features are unique to Service-Oriented Systems:

- Long-lived and concurrent transactions, not only traditional transactions which 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.

  For example, information query and flight payment need 5 minutes; while e-shopping an hour; and a complex business transaction like contracting may take days.
- Greater dynamism. Computation and communication resources may change at runtime.
- Unavailability of undo operations, most often only compensating actions that return the system to a state that is close to the initial state is available.

Furthermore, transactions may act differently when exposed to certain conditions such as logical expressions, events expressed in deadlines, and even errors in case of a faulty web service. To make sure that the integrity of data is persistent, the two transaction models used are, namely, Composite and distributed that allow smooth recovery to a previous "safe" state.

The set of emerging features mentioned earlier, which combinations of requirements are mostly coming from the areas of databases and workflows provide the basis for identifying the most relevant requirements for transactions in Service-Oriented Systems.

3.1 ACID Properties

3.1.1 Atomicity

Atomicity is the property of a transaction to either complete successfully or not at all, even in the event of partial failures.

3.1.2 Consistency

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.

3.1.3 Isolation

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.

3.1.4 Durability

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.

3.2 Transaction Behaviors

3.2.1 Rollback

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.

3.2.2 Compensating Actions

Compensating actions are executed in the event of a failure during a transaction, all changes performed before the failure should be undone.

3.2.3 Abort

Abort is the returning to the initial state in case of failure or if the user wishes so.

3.2.4 Adding Deadlines

Adding deadlines to transactions involves giving timeouts to operations.

3.2.5 Logical Expressions

Logical expressions for specifying constraints are used for giving unambiguous and semantically defined rules for guaranteeing consistency.

3.3 Transaction Models

3.3.1 Composite Transactions

Composite transactions are nested transactions. These transactions depend on the global outcome, that is, all three succeed or the whole composite transaction fails.

3.3.2 Distributed Transactions

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.

3.4 Transaction Behavior—Alternatives

3.4.1 Transaction Recovery

Transaction recovery by dynamic rebinding and dynamic recomposition at runtime is the possibility of replacing a faulty web service when the current service is not able to fulfill its promises. Dynamic recomposition is the forming of a new composition by replacing one or several services by another composition that fulfills the same function.
Imagine that the first Carrier somehow fails and is unreachable. If this happens during a transaction, then automatic rebinding with a service that offers the same service should take place. Recomposition through rebinding with a third Carrier through the Supplier is also a possibility.

3.4.2 Optimistic or Pessimistic Concurrency Control

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 resolves 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.

TRANSACTION STANDARDS AND SERVICE COMPOSITION LANGUAGES

WS-Transactions and Business Transaction Protocol (BTP) 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 and the Choreography Description Language (WS-CDL) are most widely known and adopted. WS-Transactions consist of two coordination protocols: WS-Atomic Transaction and WS-Business Activity which live in the WS-coordination framework. 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 Service-Oriented Systems with transaction properties by wrapping them with a specific coordination.

BTP 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 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 provides the infrastructure to describe cross-enterprise collaborations of web services in a choreographic way.

Consider the proposed protocols that take the transaction and the business perspective of Service-Oriented Systems with respect to the requirements. 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 "⊕" symbol, the partial satisfaction with "⊙" and no support with "Θ".

TABLE-1 Evaluation Results

Requirements	BTP	WS-AT	WS-BA	BPEL	WS-CDL
3.1.1 Atomicity	⊕	⊕	⊖	⊙	⊙
3.1.2 Consistency	⊕	⊕	⊙	⊙	⊙
3.1.3 Isolation	⊖	⊕	⊕	⊕	⊕
3.1.4 Durability	⊕	⊕	⊕	⊕	⊙
3.2.1 Rollback	⊕	⊕	⊙	⊙	⊕
3.2.2 Compensating actions	⊖	⊖	⊕	⊕	⊙
3.2.3 Abort	⊕	⊕	⊕	⊕	⊖
3.2.4 Adding deadlines	⊖	⊕	⊕	⊙	⊕
3.2.5 Logical expressions	⊖	⊖	⊖	⊕	⊕
3.3.1 Composite trans.	⊕	⊕	⊕	⊕	⊕
3.3.2 Distributed trans.	⊕	⊕	⊕	⊕	⊕
3.4.1 Trans. recovery	⊖	⊖	⊖	⊕	⊖
3.4.2 Concurrency control	⊖	⊖	⊖	⊖	⊕

WS-AT is a traditional protocol which satisfies the basic ACID properties. WS-BA, on the other hand, renounces atomicity to accommodate long-lived transactions. BTP has included confirm sets. These confirm sets let the application element choose which operations with parties in the transaction are to be canceled 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. WSAT contains simple two-phase commit protocols, WS-BA contains nonblocking protocols and BTP consists of a Sequence of small atomic transactions. As for security, WS-Security 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.

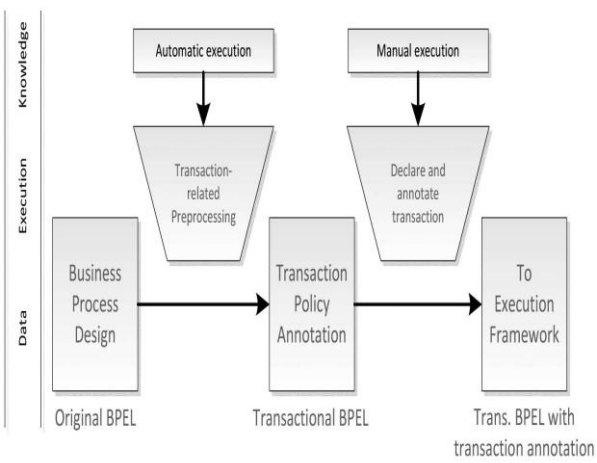
WS-AT is a very conservative business transaction model especially with respect to blocking. WS-BA is more appropriate for services, by renouncing to the concept of the two-phase commit. BTP places itself in the middle (two-phase commit is followed in a relaxed way). As for BPEL and WS-CDL, they address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

5 PROPOSALS FOR INTEGRATING TRANSACTIONS INTO BPEL

The above survey shows that there are standardized protocols for describing transactions and languages for describing processes in terms of flows of activities. The connection among these is, to say the least, very loose. The problem is that processes are described in terms of activities and roles capable of executing the activities, but semantic dependencies among these activities are not represented beyond message and flow control. It may happen that several operations from a single web service are invoked within a BPEL process, and dependencies among these operations may exist.

Our proposal consists of making the dependencies among the activities explicit via an automatic procedure and performing a restructuring step of the process, where necessary. The identified dependencies among activities can be then identified by the designer of the process as being transactions or not. In case they are, the designer will decide which kind of transactions they are and simply annotate them. The execution framework then takes care that transaction annotations are correctly managed at runtime. The need for the human design decision in the process is necessary due to the lack of semantic annotations of the BPEL processes. Only the designer can decide whether a set of activities that seem to have a dependency in the process are to be executed transaction ally or not.

Fig 3 Approach to integrating transactions into BPEL processes.



Consider Fig. 3, where data transformation goes from left to right and we distinguish three layers: the data layer at the bottom, the middle execution layer defining the data transformation, and the knowledge level indicating from where the knowledge to transform the data comes. We start with a generic business process designed to solve some business goal. An automatic processing step, which we define next, identifies dependencies among activities. These are then reviewed by an expert that decides which actually transactions are and which not. For those who qualify, he further decides what kind of transactions they are and

annotates them. For instance, some may be long running while others may be atomic ones. We remark how this is a design step performed by an expert who understands the domain, the specific process, and the consequences of choosing a transaction policy in favor of another. This step cannot be automated unless further semantic annotations are made on the BPEL. The restructured and annotated process is then ready to be sent for execution. Next the execution phase and will be handled by the execution framework. We consider the three phases of the approach individually.

5.1 Preprocessing

Preprocessing the BPEL specification is performed in two steps, namely, 1) identification and 2) resolution of transaction dependencies. In order to illustrate the two steps, we introduce an abstract model of BPEL.

5.1.1 Abstract Model of BPEL Specifications

A BPEL process specification describes the interaction between services in a specific composite web service. Its abstract model, known as *behavioral interface*, defines the behavior of a group of services by specifying constraints on the order of messages to be sent and received from a service.

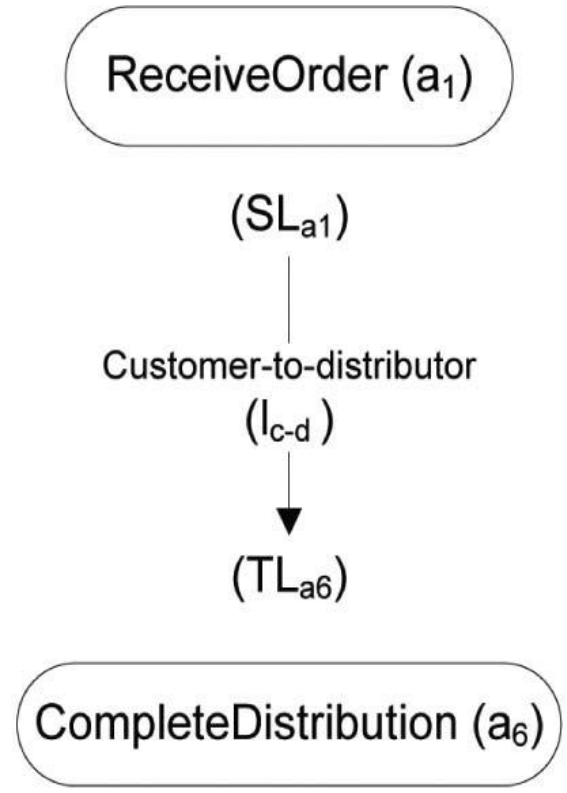
A BPEL specification 'S' is a set of activities 'A' and its associated links 'L', represented by $S = (A, L)$. The links, which are directed, define a partial ordering over the set of activities and are thus well represented as a directed graph (e.g., Fig. 4).

- An activity 'a' in A having a type represented by T_a, has the following properties:
- Name N_a.
- Operation OP_a, which is usually implemented by the web service at a specific port.
- Input variable IV_a and output variable OV_a, which specify the parameters required and produced by the OP_a, respectively.
- Set of source links SL_a and set of target links TL_a, which specify the outgoing and incoming links (transitions), respectively.

  A link l in S has a unique name N_l and is indirectly defined through two activities a_1 and a_2 which indicates not only the direction l^d of the transition, but also the conditions l^c for the transition to take place.

Furthermore, the Customer-to-distributor link $l_{c\text{-}d}$ is one of the source links of the *ReceiveOrder* activity a_1, namely, $l_{c\text{-}d} \in SLa_1$. Furthermore, $l_{c\text{-}d} \in TLa_6$, where TLa_6 is the target link of the *CompleteDistribution* activity a_6. Therefore, the link $l_{c\text{-}d}$ connects the transition between a_1 and a_6, denoted as $a_1 \xrightarrow{lc-d} a_6$. Fig. 4 provides an illustration of $a_1 \xrightarrow{lc-d} a_6$.

Figure 4: Representation of activities and the link that connects them.



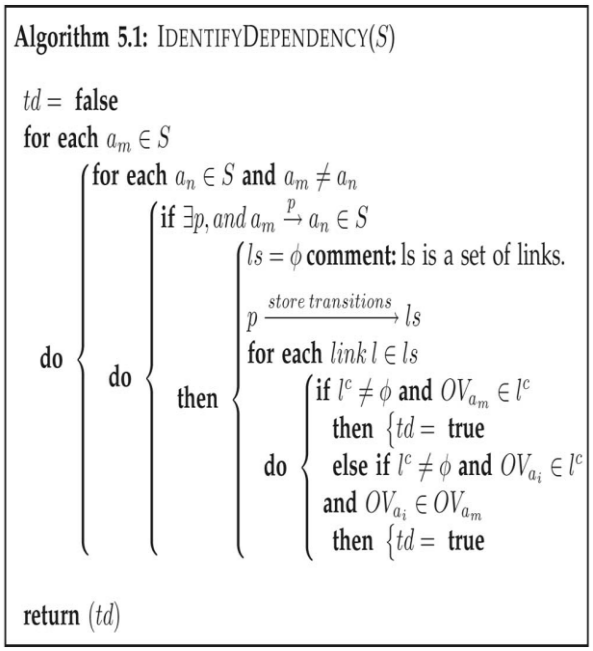
5.1.2 Dependencies Identification Algorithm

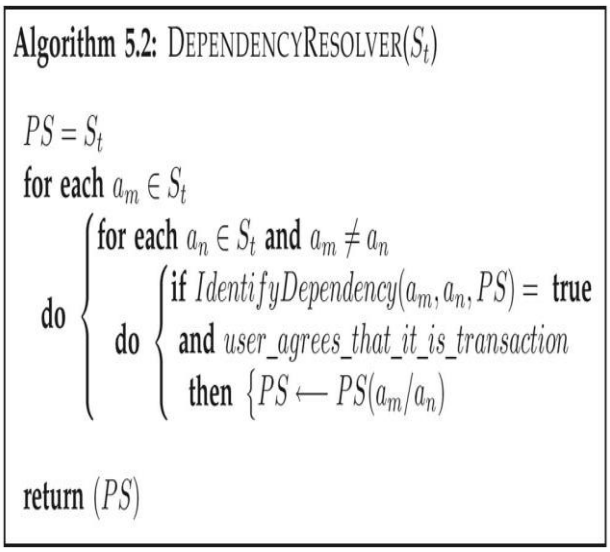
If one specifies a set of activities within a given BPEL specification S, there may exist dependencies among activities that can hinder the application of transaction management as described above. Assume that

S_t = { a_i | a_i *is a transactional activity of a transaction* t} is a transaction t specified within the BPEL specification S.

For any two activities a_m and a_n where a_m, $a_n \in S_t$ and $a_m \neq a_n$, if there exists a path $a_m \xrightarrow{lj1} \ldots \xrightarrow{ljk} a_n$ where l_{j1} and l_{jk} are some links connecting activities, we say that a_n is reachable from a_m, denoted as $a_m \overset{*}{\rightarrow} a_n$, and $\{l_{j1}, \ldots l_{jk}\}$ is a link chain of $a_m \overset{*}{\rightarrow} a_n$ denoted as $\mathbf{LC}\ \{ a_m \overset{*}{\rightarrow} a_n \}$. For any two activities a_m and a_n in a transaction S_t that are implemented by the same web service, if $a_m \overset{*}{\rightarrow} a_n$ and $OV_{am} \in l^c$ where $l \in \mathbf{LC}\ \{a_m \overset{*}{\rightarrow} a_n\}$, then a transaction dependency exists between a_m and a_n.

To identify the existence of transaction dependencies within a given BPEL specification S, we propose Algorithm 5.1.





The algorithm is a standard graph algorithm similar to those for reachable set construction. The function ***IdentifyDependency*** takes ***S*** as input and outputs a Boolean value that represents the existence of transaction dependencies. The function first creates a path ***p*** for any two activities a_m and a_n. Then, traverses the links in the link chain l_s obtained from ***p***. When a link ***l*** is detected and its transition condition l^c contains the output variable OV_{am} of the first activity a_m, or if it contains an output variable OV_{ai} which is identical to OV_{am} semantically, the algorithm stops and returns TRUE. Otherwise, it continues until all pairs of activities in S_t have been visited. Finally, if no transaction dependencies are detected, the algorithm returns FALSE.

5.1.3 Resolution of Dependencies

Once transaction dependencies are identified, it is necessary to handle them. To solve this problem, we merge the dependent activities into one transaction. Algorithm 5.2 resolves the transaction dependencies within a BPEL specification S. It employs Algorithm 5.1 to detect transaction dependencies and it asks the user for confirmation that it is indeed a transactional dependency. The output is a new
BPEL specification referred to as preprocessed BPEL where conflicts are resolved.

5.2 Declaration of Transaction Policies

Once transactions are identified and BPEL has been accordingly restructured, one needs to define the desired transactional behavior. To this end, we introduce a reference transaction policy declaration schema, shown in Fig. 7.

With this schema, one can declare the transaction policy using the following elements:

1. Trans ID is a nonzero integer, representing transactions within a business process.
2. Trans Protocol specifies a protocol for the transaction, such as WS-AtomicTransaction or WS-BusinessActivity.
3. Trans Root indicates the parent transaction identified by Trans_ID. The value 0 is used to indicate the root transaction within the business process. One can specify the hierarchy of transactions by assigning appropriate Trans_IDs and Trans_Roots.

With such a schema, one can annotate constraints or preferences to a specific activity in the BPEL specification. The annotated activity must be an invoke activity. One can separately specify the desired constraints or preferences in the design-time-info or runtime-info sections. For transaction management, we declare the transaction policies in the section of the transinfo which is embedded within the section of runtime info, since a transaction policy is a runtime constraint. Together with the other types of process information, transaction policies are stored in an XML file for use at runtime.

Figure 7: A transaction policy declaration schema.

```
<activity-info activityName=@ncname>
 <design-time-info>
      <!-- To do: define design-time information -->
 </design-time-info>
 <run-time-info>
    <!-- To do: define other run-time  information -->
    <trans-info>
          <Trans_ID> i </Trans_ID> <!-- where i is a non zero integer -->
          <Trans_Protocol> [WS_AT] | [WS_BA] </Trans_Protocol>
          <Trans_Root> Trans_ID </ Trans_Root >
    </trans-info >
 </run-time-info>
</activity-info>
```

5.3 The Execution Framework

The proposed approach transforms a generic business process into a restructured one in which transactions are identified and annotated. Now, one needs an execution framework that is richer than a simple BPEL engine. In fact, one needs to interpret the annotations, make sure that activities are executed according to the transaction conditions and also that the binding among dependent activities is consistent with the transaction semantics. To achieve this, we rely on the Service Centric System Engineering (SeCSE) platform in the context of which the current approach has been developed.

Service Centric System Engineering is a European sixth framework integrated project, whose primary goal is to create methods, tools, and techniques for system integrators and service providers and to support the cost-effective development of service-centric applications. The SeCSE service composition methodology supports the modeling of both the service interaction view and the service process view. A service integrator needs to design both the abstract flow logic and the decision logic of the process-based composition. Therefore, the SeCSE composition language allows the definition of a service composition in terms of a process and some rules that determine its dynamic behavior. Correspondingly, the flow logic can be represented by a BPEL specification, while the decision logic is defined by rules.

Based on the architecture of the SeCSE platform, we built a transaction management tool called DecTM4B. It consists of three modules, namely,

- The ***Preprocessor for Transaction Management*** is used to identify and eliminate transaction dependencies occurring in the original BPEL specification. The output is the preprocessed BPEL specification. The SeCSE platform will deal with the binding of abstract services before the BPEL engine executes the BPEL specification. The preprocessing executed by Preprocessor for T.M. happens just before the binding. Currently,ODE and ActiveBPEL are two BPEL engines supported by the SCENE platform.
- The ***Event Adapter*** maps the low-level events from the BPEL engine onto the binding-related events. The first version of SeCSE event adapter is extended to support the mapping of transaction-related events.
- The Transaction Manager is a separate component in the executor and deployed in the Mule container (Mule is a messaging platform based on ideas from Enterprise Service Bus (ESB) architectures).

The Transaction Manager consists of the following two transaction-specific components:

- *TransLog* is responsible for managing the lifecycle of transactions, such as creating transaction instances, maintaining the status of transaction instances, and destroying transaction instances. TransLog is also responsible for transferring the information among the components in the executor.

For example, it listens the transaction-related events from the Event Adapter, and it is responsible for the communication between Transaction Manager and JBoss Transaction Server.

- *PolicyOperator* retrieves the transaction policies from the XML file, and parses the transaction policies, and then maps transaction policies onto the coordination context. It provides a set of APIs which are to be called by the TransLog.

As for the implementation of transaction protocols, we rely on JBoss Transaction Server. JBoss Transaction Server is an open source implementation of WS-Coordination, WSAtomicTransaction, and WS-BusinessActivity. It provides a set of APIs to support the coordination services and transaction protocols. JBoss Transaction Server is selectedfor this purpose because it 1) is a complete, standalone, open source software tool, 2) has sufficient documentation and 3) and supports WS-Coordination and WS-Transaction.

6. CONCLUSION:

Web services are being increasingly adopted by organizations in order to run their business more effectively and efficiently. However, current technologies lack the support often required by such organizations. The success of web services lies, among other factors, in their reliability, especially when economic interests are involved. One key feature is that of being able to deal transactionally with a set of operations, but this is far from being easy, especially when the operations in the transaction come from different remote service instances.

In this paper, we highlight the key requirements of transaction management in Service-Oriented Systems and propose a novel declarative transaction

management approach for web service compositions. The key to implementing transaction management into BPEL processes 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
BPEL processes.

The proposal consists of first a preprocessing of the BPEL to identify and manage transaction dependencies among a group of activities. Then, it proceeds with the annotation with transaction policies. Finally, the interpretations of the declared transaction policy are specified as event-action condition rules to be processed at runtime.

REFERENCES

1) WS-C, "Web Services Coordination (WS-Coordination)," technical report, Arjuna Technologies Ltd., BEA Systems, Hitachi Ltd., IBM, IONA Technologies, and Microsoft, 2007.
2) WS-AT, "Web Services Atomic Transaction (WS-AtomicTransaction), Version 1.1," technical report, Arjuna Technologies Ltd., BEA Systems, Hitachi Ltd., IBM, IONA Technologies, and Microsoft, 2007.
3) WS-BA, "Web Services Business Activity Framework (WSBusinessActivity), Version 1.1," technical report, Arjuna Technologies Ltd., BEA Systems, Hitachi Ltd., IBM, IONA Technologies, and Microsoft, 2007.
4) BPEL, "Business Process Execution Language for Web Services Version 1.1," technical report, IBM, Microsoft, BEAT, SAP, and Siebel Systems, 2003.
5) C. Sun and M. Aiello, "Requirements and Evaluation of Protocols and Tools for Transaction Management in Service Centric Systems," Proc. Ann. Int'l Computer Software and Applications Conf. (COMPSAC '07), pp. 461-466, 2007.
6) A. Erradi, P. Maheshwari, and V. Tosic, "Recovery Policies for Enhancing Web Services Reliability," Proc. IEEE Int'l Conf. Web Services (ICWS '06), pp. 189-196, 2006.
7) A.Portilla, "Providing Transactional Behavior to Services Coordination," Proc. Very Large Data Bases (VLDB) PhD Workshop, vol. 170, 2006.
8) S. Loecher, "Model-Based Transaction Service Configuration for Component-Based Development," Component-Based Software Eng., pp. 302-309, 2004.
9) P.W.P.J. Grefen, "Combining Theory and Practice in Integrity Control: A Declarative Approach to the Specification of a Transaction Modification Subsystem," Proc. 19th Int'l Conf. Very Large Data Bases, R. Agrawal, S. Baker, and D.A. Bell, eds., pp. 581- 591, 1993.
10) C. Sun and M. Aiello, "Requirements and Evaluation of Protocols and Tools for Transaction Management in Service Centric Systems," Proc. Ann. Int'l Computer Software and Applications Conf. (COMPSAC '07), pp. 461-466, 2007.
11) A. Lazovik, M. Aiello, and M. Papazoglou, "Planning and Monitoring the Execution of Web Service Requests," Int'l J. Digital Libraries, vol. 6, no. 3, pp. 235-246, 2006.
12) M. Aiello and A. Lazovik, "Monitoring Assertion-Based Business Process," Int'l J. Cooperative Information Systems, vol. 15, no. 3, pp. 359-390, 2006.
13) B. Haugen and T. Fletcher, "Multi-Party Electronic Business Transactions. Version 1.1," technical report, UN, 2002.

AUTHOR BIOGRAPHIES:

R.Soujanya received her B.Tech, Degree in Computer science and information technology from G.Pulla Reddy Engineering College, Kurnool India, in 2010. She is pursuing her M.Tech in Software Engineering in Sri Kottam Tulasi Reddy Memorial College, Kondair India. Her area of interest is in the field of Service oriented computing, Wireless Sensor Networks and Software Testing.

Mr.M.Venkateshwarlu Associate Professor, completed B.Tech in Computer Science & Engineering in Sri Kalahastheeshwara Institute of Technology, Sri Kalahasthi, and M.Tech in Computer Science & Engineering in Sri Kottam Tulasi Reddy Memorial College of Engg, Affiliated to JNTU Hyderabad. His interested areas are Wireless Sensor Networks, Artificial Intelligence. He attended two international conferences on MANET`s and one national level conference on Networks. He attended Mission 10x program conducted by Wipro. Presently he was acting as The Convener of R&D cell in SKTRMCE. Kondair, Mahaboob Nagar (Dt). He had total 07 years of teaching experience. Currently he was working as Associate Professor in CSE Department in KTM College of Engineering,