A Survey of Transactional Issues for Web Service Composition and Recovery

Le Gao

Department of Computer Science
Texas Tech University
Lubbock, Texas, USA, 79409
E-mail: le.gao@ttu.edu

Susan D. Urban*

Department of Industrial Engineering
Texas Tech University
Lubbock, Texas, USA, 79409
E-mail: susan.urban@ttu.edu
*Corresponding author

Janani Ramachandran

Department of Computer Science
Texas Tech University
Lubbock, Texas, USA, 79409
E-mail: janani.ramachandran@ttu.edu

Abstract:

This paper presents a survey of relevant transactional and recovery issues for the development of processes that are composed of web services in a service-oriented architecture (SOA). A process in an SOA is not a traditional ACID transaction due to the loosely-coupled, autonomous, and heterogeneous nature of the execution environment. As a result, processes that are composed of web services can pose challenges for data consistency in the context of concurrent processes that are accessing shared data. This paper first outlines past research on advanced transaction models and transactional workflows that has established the framework for coordination and recovery techniques associated with web service composition. The standards that have evolved to support web service composition and a coordinated commit process among web services are then presented. The paper then elaborates on research projects that address data consistency issues for web service composition, outlining relaxed locking techniques, data dependency analysis techniques, and other modularization techniques for addressing user-defined correctness, flexible recovery actions, and cross-cutting concerns. Failure recovery strategies for Web Services are also addressed, describing how current research builds on the foundation provided by advanced transaction models to perform recovery for processes composed of Web Services. The overview of failure recovery strategies also includes self-healing mechanisms and

checkpointing systems for service execution. The paper concludes by outlining challenges for future research on web service composition and recovery.

Keywords:

Reference to this paper should be made as follows: Le Gao, Susan Urban and Janani Ramachandran. (xxxx) 'A Survey of Transactional Issues for Web Service Composition and Recovery', *Int. J. Web and Grid Services*, Vol. x, No. x, pp.xxx–xxx.

Biographical notes: Le Gao received his B.S. degree in computer science in 2004 from Nanjing University of Aeronautics and Astronautics, China. He is currently a Ph.D. candidate in the Department of Computer Science at Texas Tech University. Before coming to Texas Tech University, he was a database engineer at China Mobile Corporation from 2006-2007. He also worked at China Telecom Corporation as a database operator from 2004-2006.

Susan D. Urban received the B.S, M.S., and Ph.D. degrees in computer science in 1976, 1980, and 1987, respectively, from the University of Louisiana at Lafayette. She is currently a Professor in the Department of Industrial Engineering at Texas Tech University. She was previously at Arizona State University from 1989-2007, where she currently holds the status of Emeritus Professor. She was also an Assistant Professor at the University of Miami from 1987-1989. Her research addresses integrated techniques for event, rule, and transaction processing to address data consistency and active behavior in distributed, data-centric applications.

Janani Ramachandran received the B.S. degree in computer science in 2009 from Anna University, India. She is currently an M.S. student in the Department of Computer Science at Texas Tech University investigating flexible, event-driven exception handling for processes composed of web services.

1 Introduction

With the development of the Internet, Web Services and service-oriented computing are becoming more widely used to support processes and data exchange in business-to-business integration as well as business and scientific-oriented workflows. In a traditional, data-oriented, distributed computing environment, a distributed transaction is used to provide certain correctness guarantees about the execution of a transaction over distributed data. In particular, a traditional, distributed transaction provides all-or-nothing behavior by using the two-phase (2PC) commit protocol to support atomicity, consistency, isolation, and durability (ACID) properties (Elmasri and Navathe, 2011). A process in a service-oriented

architecture (SOA) (Singh and Huhns, 2005), however, is not a traditional ACID transaction due to the loosely-coupled, autonomous, and heterogeneous nature of the execution environment.

In an SOA, a service is highly independent of the context and the state of other services invoked by a process. Since a service is autonomous and platform-independent, the commit of a service execution is controlled by the host execution environment of the service instead of the global process. Therefore, processes composed of web services do not generally execute as transactions that conform to the concept of serializability. Concurrently running processes may access or modify the same data through independent services without the isolation property in between service executions. As a result, dirty reads or dirty writes may occur from the global process perspective. If a process fails, the recovery of the process may affect the data consistency in other concurrent processes that have accessed the same data. Therefore, the recovery of a failed process is not enough to maintain data consistency from a global perspective due to potential dirty reads or writes.

As an example, consider the scenario shown in Figure 1, where three processes are running concurrently in an SOA. Process_1 is initiated by agent_1, while process_2 and process_3 are both controlled by agent_2. In Figure 1, operation_{32} and operation_{23} each invoke service_2. If process_3 fails at operation_{34} and recovers operation_{32}, then process_2 might be affected due to the potential dirty read/write problem. The dirty read/write issue might also happen between processes that are controlled by different agents. For example, the recovery of process_2 controlled by agent_2 might affect the correctness of process_1 controlled by agent_1, since operation_{21} and operation_{12} both execute at service_1 and potentially access the same data. In an SOA, it is first of all a challenge to provide flexible recovery techniques that can respond to service errors, exceptions, and interruptions in a manner that preserves the data constraints of a single process and attempts to keep the process running. It is also a challenge to analyze data dependencies between concurrently running processes to determine how the data changes caused by the recovery of one process can possibly affect other processes that have either read or written data modified by the services of the failed process.

This paper presents a survey of relevant transactional and recovery issues for the development of processes that are composed of web services, with a focus on data consistency in the context of concurrent processes. The survey begins in Section 2 with an overview of foundational research in the area of advanced transaction models and transactional workflows. Current work with service composition shares many of the same characteristics as that of seminal research on advanced transaction models and workflows. Much of this original research was conducted in the context of long-running transactions, where consistency techniques such as serializability and the isolation property were generally unattainable. As a result, many of the recovery techniques for service composition have been derived from this earlier work with long-running transactions.

Section 3 then provides an overview of the most relevant standards that have evolved to provide some form of coordinated commit process among the services involved in a process. In particular, we outline WS-Coordination as a framework for a coordination service, as well as WS-Transactions for achieving 2PC and WS-Business Activity for achieving a more relaxed execution of long-running business

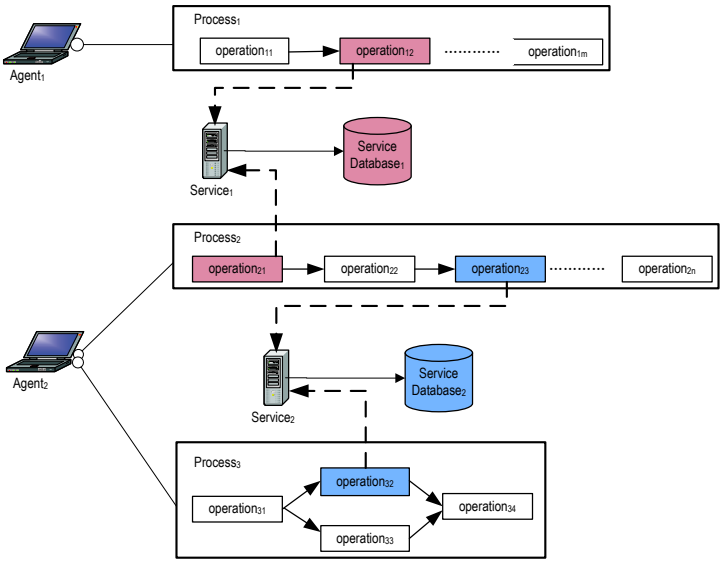

Figure 1 Process Execution in an SOA

activities that cross vendor boundaries. We also provide an overview of WS-BPEL 2.0 as a standard for web service composition, summarizing fault, compensation, and termination handlers for dealing with recovery issues.

Section 4 presents research projects that address data consistency issues for web service composition, outlining relaxed locking techniques, data dependency analysis techniques, and other modularization techniques for addressing user-defined correctness, flexible recovery actions, and cross-cutting concerns. Section 5 then elaborates on failure recovery strategies for Web Services, describing current research that builds on the foundation provided by advanced transaction models to perform recovery for processes composed of Web Services. We also outline current work with self-healing mechanisms and checkpointing systems for service execution. The paper concludes in Section 6 with a summary and discussion of future research directions.

2 A Historical Perspective of Advanced Transaction and Workflow Models

Current techniques for handling the transactional aspects of web service composition and recovery have been influenced by past research with advanced transaction and workflow models. To establish the context for current research on service composition and recovery, this section presents relevant background on advanced transaction models and transactional workflows.

2.1 Advanced Transaction Models

The traditional notion of transactions with ACID properties is too restrictive for the types of complex transactional activities that occur in distributed applications,

primarily because locking resources during the entire execution period is not applicable for Long Running Transactions (LRTs) that require relaxed atomicity and isolation (Cichocki, 1998). Advanced transaction models (ATMs), such as Sagas, the Nested Transaction Model, the Multi-level Transaction Model and the Flexible Transaction Model, have been defined to better support LRTs in a distributed environment (Rolf *et al.*, 1997; Elmagarmid, 1992).

The notion of a Saga (Garcia-Molina and Salem, 1987) was proposed in 1987 as a base model for long-running activities. A Saga consists of many ordered smaller tasks that conform to ACID properties, where these tasks execute in a sequence. The Saga relaxes the requirement of the entire transaction as an atomic action by releasing resources before the transaction completes without sacrificing the consistency of the database. A compensator is created with each task in a Saga. The compensator is an execution that can logically undo the results of the task. When a Saga needs to be aborted, the system aborts the current active task and executes the compensators for each task in reverse order to backward recover the entire Saga process. As shown in Figure 2, when the saga fails at $task_3$, $compensation_2$ and $compensation_1$ will be executed to logically undo $task_2$ and $task_1$. Compensation as a recovery technique is an important contribution of the Saga model that has had a significant influence over current recovery techniques for web service composition.

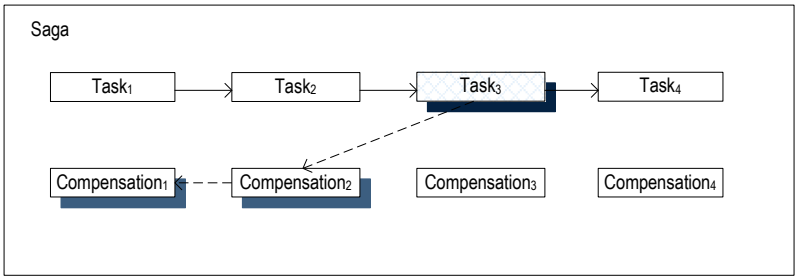

Figure 2 Recovery of a Failed Saga Process

Nested transactions provide another foundational, introducing hierarchical transaction structure together with the concepts of contingency and non-vital sub-transactions (Moss, 1985, 1987). In the nested transaction model, a parent transaction can be decomposed into sub-transactions (or child transactions), where sub-transactions can be further decomposed in a similar manner. Top-level transactions of a transaction hierarchy support ACID properties. A transaction cannot commit until all of its child transactions have committed. Locks at the child level are also inherited at the parent level. If a child transaction fails, the parent transaction can ignore the failure and thus treat the child as non-vital. The parent can also decide to execute a contingent sub-transaction as an alternative action.

A more relaxed concurrent execution model that provides an extension of the nested model is the multi-level transaction model introduced in (Weikum, 1991), where a transaction is decomposed into a nested set of sub-transactions at different levels and then each sub-transaction can commit on its own before

the whole transaction commits. A sub-transaction can create its sub-transactions at the next level as child sub-transactions. The commit of a parent transaction must wait until the child sub-transactions commit. In case of abort at the parent level, the committed sub-transactions will run their own compensators to perform "undo" actions.

A flexible transaction model which is suitable for a multidatabase environment was presented in (Elmagarmid *et al.*, 1990). A flexible transaction defines a set of equivalent alternative sub-transactions. The flexible transaction model relaxes global atomicity by allowing the transaction designer to define a set of acceptable termination states. Therefore the successful execution of a transaction will be the successful execution of a set of sub-transactions or its alternatives. In the flexible transaction model, an acceptable state is also used to decide whether to commit, abort, or compensate a sub-transaction. The flexible transaction model, therefore, introduces the concept of user-defined correctness for transaction execution in multidatabase environments.

These advanced transaction models, as well as other models outlined in (Elmagarmid, 1992), relax the ACID properties of traditional transaction models to better support LRTs and to provide a theoretical basis for further study of complex distributed transaction issues, such as failure atomicity, isolation, consistency, and concurrency control. These models have also defined important concepts, such as compensation, contingency, non-vital transactions, and user-defined correcteness, that are important to current approaches for web service composition and recovery.

2.2 Transactional Workflow

The term transactional workflow was introduced to recognize the relevance of transactions to workflow activity that does not fully support ACID properties. Transactional workflows contain the coordinated execution of multiple related tasks that support access to heterogeneous, autonomous, and distributed data through the use of selected transactional properties for individual tasks or entire workflows (Worah and Sheth, 1997). Transactional workflows are usually non-atomic and long-lived processes, containing a set of tasks executed at different sites. Transactional workflows require externalizing intermediate results, while at the same time providing concurrency control, consistency guarantees, and a failure recovery mechanism for a multi-user, multi-workflow environment. Concepts such as rollback, compensation, forward recovery, and logging have been used to achieve workflow failure recovery in several projects.

The ConTract Model provides a classic example of work with transactional workflows (Wächter and Reuter, 1991). The model was originally proposed for use in applications such as office automation and manufacturing control. A ConTract model consists of a set of predefined actions that conform to ACID properties called steps and an explicitly specified execution plan called a script. The ConTract Model provides compensation for backward recovery, and basic constraint checking through the specification of pre-conditions or post-conditions for steps. After the execution of each step, the ConTract Model will release locks and, if failure occurs, the ConTract Model will logically recover completed steps. A unique contribution of the ConTract Model is forward recoverability, defining the ability of

an execution to do a partial recovery when failure occurs and to continue forward execution from the point of failure. The ConTract model also emphasizes user-defined consistency through the use of invariants, or consistency constraints, on the database for the purpose of concurrency control.

Eder and Liebhart (1995) introduced the workflow activity model (WAMO). WAMO supports modeling complex business processes in a simple and reliable way. In WAMO, a complex business process is composed of a set of smaller work units, known as activities. In this model, a workflow consists of three basic units: activity, form and agent. An activity represents the abstract description of a work unit in the business process. A data repository or container to store relevant data is called a form. An agent is a processing entity to perform the execution of activities. An activity may consist of multiple other activities as its steps. Furthermore, activities are reusable by other activities so that new processes can be composed of existing activities. In WAMO, the state after each execution is used to control activities of the workflow.

The Units of Work (UOW) model was proposed in (Bennett *et al.*, 2000), supporting long running business processes in a distributed environment. The UOW model is an advanced nested transaction model that enables concurrent access to shared data without locking resources. In this model, a nested LRT is represented by the UOW object and the non-UOW-aware base object provides business functionalities. The model takes the base objects, creates versions that are associated with UOWs, and maps method invocation under a given UOW context onto the set of objects associated with the UOW. In this model, an LRT is represented as a tree structure whose nodes are UOWs, each of which is a nestable LRT. The isolation property is guaranteed by three visibility rules: 1) the state of all the objects in a parent UOW is visible to all child UOWs of that parent UOW, 2) the state change of a child UOW is visible to the parent after the child commits, and 3) a child state change is not visible to its sibling before it commits.

The Correct and Reliable Execution of Workflows (CREW) project (Karnath and Ramamritham, 1998) introduced correctness requirements and other defined constraints into transactional workflows. A workflow in CREW includes multiple steps. The completion of previous steps will trigger the execution of the next steps. The occurrence of specific events can also trigger the execution of specified steps. The rules, events or conditions are used to manage the execution of workflows. Handling of failures to eliminate unnecessary compensations and re-execution of steps are also supported in CREW. If the execution of a step fails, complete compensation and re-execution, or partial compensation and incremental re-execution is invoked to recover the error. Therefore, CREW provides a more dynamic workflow by the use of rules and the mechanisms for handling failures and exceptions.

The METEOR model (Wodtke *et al.*, 1996) combines many features such as two-phase commit (2PC) coordination, error handling, and failure recovery from other transactional workflows models. A METEOR model includes four components: processing entities and their interfaces, tasks, task managers and workflow schedulers. A processing entity is responsible for executing a task. A task is a basic execution agent that performs operations. The task manager takes control of each task. The workflow scheduler is responsible for coordinating the execution of tasks. METEOR uses a three-layer error model to handle workflow

errors, categorizing runtime errors as task errors, task manager errors, or workflow engine errors. These errors can be handled automatically or by human agents by different methods introduced in METEOR.

Although the above advanced transaction and workflow models have provided techniques that support access to heterogeneous, autonomous, and distributed data among multiple related tasks, these models do not fully support the isolation, failure atomicity, timed constraints, and liveness requirements of distributed transactional workflows (Kuo *et al.*, 2002). Many of these models make assumptions about the transaction semantics of sub-transactions. Furthermore, workflow activities, as well as web service composition, often cross organizational boundaries and involve human intervention, complex programming control structures, and diverse service environments. As in the work with transactional workflows, more comprehensive and suitable recovery techniques are needed to support the transactional needs of web service compositions. The following sections examine some of the standards and research projects that have addressed these issues in the context of web service composition.

3 Standards for Web Service Transactions and Composition

In an SOA, standard business functionality is provided by well-defined, self-contained software modules, known as services. In (Papazoglou and Georgakopoulos, 2003), a service is defined to be an exposed piece of functionality that: 1) is self-contained and maintains its own state, 2) is platform-independent, and 3) can be dynamically located, invoked and (re-)combined. The main purpose of a service in an SOA is to represent a reusable unit of work. Services must be used, however, in a manner that maintains application constraints and consistency with respect to concurrent data access. Section 3.1 first summarizes standards associated with transactional issues for web services. A brief summary of the Business Process Execution Language and the features provided for service composition and recovery is then presented in Section 3.2.

3.1 Web Service Specifications

WS-Coordination (Cabrera *et al.*, 2002a) describes an extensible framework for providing protocols that coordinate the actions of distributed applications. Such coordination protocols are used to support a number of applications, including those that need to reach consistent agreement on the outcome of distributed activities. WS-Coordination describes a framework for a coordination service (or coordinator) which consists of these component services. The first component is an activation service with an operation that enables an application to create a coordination instance or context. The second component is a registration service with an operation that enables an application to register for coordination protocols. The third component is a coordination type-specific set of coordination protocols. The coordination protocols that can be defined in this framework can accommodate a wide variety of activities, including protocols for simple short-lived operations and protocols for complex long-lived business activities.

Atomic Transactions defined in WS-Transaction (Cabrera *et al.*, 2002b) build on WS-Coordination, which defines an activation and a registration service. WS-Transaction has two characteristics. One is the all or nothing property, where the actions taken prior to commit are only tentative. The other is atomic transactions that require a high level of trust between participants and are short in duration. WS-Transaction usually uses Two-Phase commit (2PC) to guarantee ACID properties. The 2PC protocol coordinates registered participants to reach a commit or abort decision, and ensures that all participants are informed of the final result. Volatile 2PC involves participants managing volatile resources such as a cache. Durable 2PC involves participants managing durable resources such as a database. Based on each protocol's registered participants, the coordinator begins with Volatile 2PC, then proceeds through Durable 2PC.

The WS-Business Activity (Cabrera *et al.*, 2005) specification defines protocols that enable existing business processes and workflow systems to wrap their proprietary mechanisms and interoperate across trust boundaries and different vendor implementations. Usually WS-Business Activity provides long-running, compensation-based transaction protocols, where a business activity may consume many resources over a long duration. There may be a significant number of atomic transactions involved. Individual tasks within a business activity can be seen prior to the completion of the business activity since their results may have an impact outside of the computer system. Responding to a request may take a very long time. Human approval, assembly, manufacturing, or delivery may have to take place before a response can be sent. In the case where a business exception requires an activity to be logically undone, abort is typically not sufficient. Exception handling mechanisms may require business logic, for example in the form of a compensation task, to reverse the effects of a previously completed task. Participants in a business activity may be in different domains of trust, where all trust relationships are established explicitly.

In contrast to WS-Transaction, the model of WS-Business activity has several distinct differences. The participant list is dynamic and a participant may exit the protocol at any time without waiting for the outcome of the protocol. WS-Business activity allows a participant task within a business activity to specify its outcome directly without waiting for solicitation. Participants in a coordinated business activity can also perform tentative operations as a normal part of the activity. There are two coordination protocols for business activities. One is BusinessAgreementWithParticipantCompletion protocol. A participant registers for this protocol with its coordinator, so that its coordinator can manage it. A participant must know when it has completed all work for a business activity. The other is BusinessAgreementWithCoordinatorCompletion protocol, which means a participant registers for this protocol with its coordinator, so that its coordinator can manage it. A participant relies on its coordinator to tell it when it has received all requests to perform work within the business activity. The main difference between the two protocols is that one executes by itself and the other one executes by a coordinator. The work in (Riegen *et al.*, 2010) provides an example of using WS-Business Activity to provide transactional support for distributed processes in service-oriented environments.

3.2 Web Services Business Process Execution Language

The Web Services Business Process Execution Language (WS-BPEL 2.0) (Jordan *et al.*, 2007) provides a standard language to specify business processes that are composed of Web services and to expose the process as a Web Service. The main concepts in WS-BPEL include processes, partner links, properties, and correlation, as well as basic and structured activities scopes. WS-BPEL defines a model and grammar for the description of the behavior of business processes on the basis of interactions between the processes and their partners. A WS-BPEL process defines the state and business logic required for the coordination of multiple service interactions with these partners to achieve business goals. A BPEL process by itself is a web service representing the business process in XML, building on WS-Standards such as WSDL, SOAP and UDDI.

BPEL defines two classes of activities: basic and structured. These activities are defined to perform the business process logic. Basic activities describe the fundamental steps of the process logic, including constructs such as $< invoke >$ to call web services, $< receive >$ to receive messages, $< reply >$ to respond to requests, $< assign >$ for operate on values of variables, $< throw >$ to indicate internal faults, $< wait >$ to specify a delay, $< empty >$ to do nothing, $< exit >$ to end the instance of business process immediately, and $< rethrow >$ to rethrow the faults caught by fault handlers. These basic activities can be used together to describe more complex tasks using structured activities. Structured activities represent the control-flow logic of the process and can consist of other basic or structured activities. Some of the structured activities are: $< sequence >$ to perform the activities contained sequentially, $< if >$ to perform conditional behavior, $< while >$ and $< repeatUntil >$ for repeated execution of contained activity, $< pick >$ to execute the activity of a particular chosen event after its occurrence, and $< flow >$ for parallelism and synchronization of activities.

BPEL provides fault, compensation and termination handlers to handle execution exceptions. All three handlers are associated with scopes. A fault handler aims to correct the error in a scope such that a process can continue running or invoke an alternative process. A compensation handler is used to compensate a completed scope. A termination handler aborts a running scope. When an error occurs, all running activities in the scope in which the error occurs will be first terminated. If the activity is a non-scope activity, it is simply aborted. If the activity is a scope, the associated termination handler is activated. When all running activities have been terminated, the fault handler associated with the scope in which the error occurs takes place. The fault handler will invoke compensation handlers to compensate all its nested completed scopes. In BPEL, the compensation procedure in a scope follows the reverse order of enclosed scope completion. However, the dependency between enclosed scopes will potentially complicate the compensation order. Because the default exception handling mechanism in WS-BPEL may activate handlers at different levels when scopes at different levels are being recovered, the recovery procedure in WS-BPEL can be difficult to understand (Khalaf *et al.*, 2009).

4 Data Consistency Approaches for Web Service Composition

Whereas the standards described in Section 3 provide protocols that allow Web Services to participate in more traditional, coordinated completion frameworks, current research is investigating more advanced techniques for maintaining data consistency among concurrent processes that execute in an SOA. This section reviews some of the relevant techniques that have been investigated for maintaining data consistency in service execution. Section 4.1 address techniques that support variations in transaction semantics capabilities as well as relaxed locking techniques. Section 4.2 then outlines techniques for data dependency analysis among concurrently executing processes. The assurance point approach for nested, hierarchical service composition together logical and physical checkpoints with user-defined correctness conditions and rule-driven recovery actions is described in Section 4.3. Section 4.4 describes functional modularity for service composition using aspect-oriented concepts.

4.1 Relaxed Semantics and Locking Techniques

Mikalsen *et al.* (2002) introduced a new Web Service Transaction (WSTx) framework, called transactional attitudes, to support the issue of transactional reliability in web service composition. In the WSTx framework, transactional attitudes are used to allow web service providers to declare their individual transactional capabilities and semantics and to allow web service clients to declare their transactional requirements. There are two types of attitudes defined in WSTx framework: Provider Transactional Attitudes (PTAs) and Client Transactional Attitudes (CTAs). PTAs are used for web service providers to explicitly describe their specific transactional behavior, while CTAs allow the clients to describe their expectations and outcome acceptance criteria explicitly. Each client executes one or more actions within the scope of a web transaction, where each action represents a provider transaction that executes within the context of the larger web transaction. The WSTx framework provides reliability during execution by using both PTAs and CTAs to define attitudes for web services transaction compositions. Also, middleware which acts as an intermediary between a client and multiple web service providers has been developed.

The Tentative Hold technique (Limthanmaphon and Zhang, 2004) provides an approach to avoid traditional locking of data to support isolation. This approach allows tentative, non-blocking holds on required resources. The resource owners, on receiving requests, grant non-blocking reservations on their resources. At the same time, they maintain control over their resources and allow several clients to place their requests for resources. This technique minimizes the need for cancellation of transactions by providing the clients with the correct and up-to-date data. This is done by maintaining track of the hold through several states. The states of a hold are Responding (initial state when an application sends a request for a hold), In Process (intermediate state which indicates that a hold request has been received), Active (state reached when the requested hold has been granted), and Inactive (state that indicates that the tentative hold is no longer valid). There are tentative hold coordinators at both the client (client coordinator) and resource owner sides (resource coordinator). The client coordinator determines the status

of previously granted tentative holds and allows a client to request holds from resource owners for specific resources, query the status of existing holds owned by it, cancel existing holds owned, query logged activities, and request to modify existing holds. The resource coordinator first checks the status of previously granted holds and verifies their expiration times, allowing a resource owner to query existing holds, cancel existing holds, and query activities. The resource coordinator is also responsible for notifying affected client coordinators when a relevant resource becomes unavailable. Thus this approach provides a non-blocking mechanism of temporarily holding resources.

Another similar method to temporarily lock data in a concurrent environment, is the reservation-based approach (Zhao *et al.*, 2005). This approach reserves resources that meet the criteria of what the web service has requested. In this protocol, each task within a business activity is divided into two steps. The first step is to reserve resources based on business logic. Basically, the reservation is a contract between the client and the resource provider. To maximize the execution concurrency in the system, a 'fee' is associated with each reservation proportional to the duration of the reservation, which discourages the application to reserve the same resources for an extended period of time. In the second step, the reservation is either confirmed or cancelled according to the business rules. Because the resource that the application requests is reserved in the first step, the application has the choice and freedom to decide about either continued execution or backtracking. A two-phase protocol is used to coordinate the different tasks within a business activity. In the first phase, the client coordinator sends reservation requests to all the participants. The confirmation or cancellation of the reservations is decided by the coordinator at the end of the first phase. In the second phase, the confirmation or cancellation requests are sent to the corresponding participants. If a participant has accepted a reservation, it must be committed to the reserved resource unless the coordinator cancels the reservation. In traditional transactions, any of the participants have the right to rollback or abort the entire transaction. In the reservation-based coordination protocol, however, only the coordinator can determine this.

The Promises approach was proposed in (Jang *et al.*, 2007) to support the isolation property in web service composition. The goal of the Promises approach is to ensure that certain values are not overwritten or changed by concurrently executing web services. A promise is an agreement between a client application and a service or promise maker. A promise assures the client that some set of conditions (predicates) will be maintained over a set of resources for a specific duration of time, as requested by the client. Instead of locking data, the approach defines the promise maker to be a promise manager that records promises. The main functionality of the promise manager is to address promise making, check on resource availability, and also ensure that promises are not violated during the specific time period. Client applications send the promise manager information in the form of predicates about the resources they want in order to complete successfully. These predicates are Boolean expressions over the resources. The request for a promise will be examined by the promise manager, which will either grant or reject the request. Once a promise request is granted, the client application is isolated from the effects of concurrent execution and can complete successfully. One method that has been used to implement promises is the concept

of soft locks. This method uses a field in the database record to indicate whether an item has been allocated already for a client or not. When an application requests the same resource, this field is read to determine availability of the resources. Promises are a weaker form of locking, but do allow other web services to access the data so that any wait is avoided.

4.2 Data Dependency Analysis

An alternative to relaxed locking techniques is the concept of data dependency analysis. The work of (Xiao, 2006; Xiao and Urban, 2008a) provides a formal definition of a process dependency model, defining read and write dependencies at the operation and process level. As illustrated in Figure 1, a failed process may affect the correctness of all other processes that are dependent on the failed process. By using the process dependency model, a set of processes that are data dependent on the failed process can be formed. In (Xiao, 2006; Xiao and Urban, 2008b,a, 2012), process interference rules (PIRs) are used to test user-defined conditions that determine if a dependent process should continue running or invoke its own recovery procedures. Xiao's process dependency model therefore provides a more optimistic approach to data access, also relying on user-defined conditions to determine whether a dependent process needs to invoke recovery actions. The work in (Xiao and Urban, 2012) elaborates on the algorithms for constructing process dependency graphs and the coordination of the recovery procedures of the service composition model with the execution of process interference rules. A simulation and evaluation of process dependency graph construction and the concurrent process recovery algorithm is also presented.

Xiao's process dependency model is based on a data dependency analysis technique that makes use of a concept known as Delta-Enabled Grid Services (DEGS) (Blake, 2005; Urban *et al.*, 2009b). A DEGS is a Grid Service that has been enhanced with an interface that provides access to the incremental data changes, or deltas, that are associated with service execution in the context of globally executing processes. Deltas captured over the source database are stored in a delta repository that is local to the service. Deltas are then generated as a stream of XML data from the delta repository and communicated to the delta event processor of the DeltaGrid environment. A centralized Process History Capture System (PHCS) (Xiao *et al.*, 2006; Urban *et al.*, 2009b) has been developed to receive deltas from different DEGSs. A complete global delta object schedule can be formed according to the timestamps of the deltas. Therefore, the global delta object schedule can be used to form the process dependent set for the purpose of triggering PIRs. Other techniques to deal with heterogeneity, autonomy, distribution and high volume of data in Grid services can be found in (Taniar and Goel, 2007; Taniar *et al.*, 2008).

One of the disadvantages of the data dependency analysis technique is that the analysis is performed by forwarding all deltas to a centralized location, thus creating a bottleneck for the analysis process. The work in (Liu, 2009; Urban *et al.*, 2011a) has developed an approach that performs decentralized data dependency analysis among concurrently executing processes through Process Execution Agents (PEXAs). PEXAs are responsible for controlling the execution of processes that are composed of web services. PEXAs are associated with

specific distributed sites and are also responsible for capturing and exchanging information with other PEXAs about the data changes that occur at those sites in the context of service executions through the construction of distributed process dependency graphs (Liu, 2009; Urban *et al.*, 2009a, 2011a). The challenge with building distributed process dependency graphs is discovering the global, hidden dependencies that are created by having each PEXA maintain its own local delta object schedule. Link objects and other runtime control information is used to discover hidden dependencies as well as global cycles in the construction of the graphs. Distributed graphs are also used to propagate recovery procedures among distributed PEXAs.

The decentralized data dependency analysis approach represents a new way of integrating existing transaction processing theories with execution platforms that can be used to address data consistency issues for concurrent process execution in service-oriented environments, providing more dynamic and intelligent ways of monitoring failures, detecting dependencies, and responding to failures and exceptional conditions.

4.3 The Assurance Point System

In addition to decentralized data dependency analysis, the work in (Xiao, 2006; Xiao and Urban, 2008a; Urban *et al.*, 2009b; Xiao and Urban, 2012) has also led to the development of the Assurance Point System for service composition and recovery, with a specific focus on user-defined data consistency and recovery techniques for processes composed of web services. An abstract service composition model was defined as a hierarchical service composition structure, where a process is composed of atomic and/or composite groups (Xiao and Urban, 2009). An atomic group is a service execution with optional compensation and contingency procedures. A composite group is composed of two or more atomic and/or composite groups and can also have optional compensation and contingency procedures. The work in (Xiao, 2006; Xiao and Urban, 2009) presents the full specification of the model using state diagrams and algorithms to define the semantics of compensation and contingency in the recovery process.

The service composition and recovery model in (Xiao and Urban, 2009) was extended with the concept of Assurance Points (APs) and integration rules to provide a more flexible way of checking constraints and responding to execution failures (Shrestha, 2010; Urban *et al.*, 2010, 2011d). An AP is a combined logical and physical checkpoint. As a physical checkpoint, an AP provides a way to store data at critical points in the execution of a process. Unlike past work with checkpointing, such as that of (Luo, 2000; Dialani *et al.*, 2002) where checkpoints are used to port an execution to a different platform as part of fault tolerant architectures, APs support user-defined consistency checking and rollback points that can be used to maximize forward recovery options when failures occur. In particular, an AP provides an execution milestone that interacts with integration rules. Failure of a pre or post-condition or the failure of a service execution can invoke several different forms of recovery, including backward recovery of the entire process, retry attempts, or execution of contingent procedures. The unique aspect of APs is that they provide intermediate rollback points when failures occur that allow a process to be compensated to a specific AP for the purpose

of rechecking pre-conditions before retry attempts. APs also support a dynamic backward recovery process, known as cascaded contingency, for hierarchically nested processes in an attempt to recover to a previous AP that can be used to invoke contingent procedures or alternate execution paths for failure of a nested process. A Petri Net formalization of the AP model and recovery techniques appears in (Urban *et al.*, 2011d).

Invariant rules and application exception rules are additional rule forms that have been defined for use with the AP approach (Urban *et al.*, 2011b). Whereas integration rules can be used to check data conditions at certain points in process execution, invariant rules are activated with a starting AP, deactivated with an ending AP, and monitor the data of the invariant condition in between APs using DEGS (Urban *et al.*, 2011c). An invariant therefore allows a process to declare data conditions that are critical to the execution of the process, but to allow multiple processes to access the same data in an optimistic fashion. When critical data conditions are violated, as detected by the DEGS capability, recovery conditions can be invoked. Application exception rules provide a case-based rule structure that can be used to interrupt the execution of a process in response to exceptional conditions and to respond to exceptions in different ways depending on the state of the executing process (Urban *et al.*, 2011b; Ramachandran, 2011). The conditions and actions that are executed in response to an external application event provide a way to check different user-defined data conditions and to take different recovery actions depending on the most recent AP that has been executed in a process.

4.4 Functional Modularization With Aspect-Oriented Techniques

Current workflow languages do not support modularization of concerns, such as data constraints and security, that span across process boundaries. The code of such crosscutting concerns is often spread across several workflows and is thus tangled with the code addressing other concerns. This has led to complex workflow process specifications that are difficult to comprehend, maintain, modify, and reuse. To address these issues, (Charfi and Mezini, 2006) proposed the use of aspect-orientation concepts in workflow languages. Aspect-oriented workflow languages provide crosscutting modularity by the use of concepts such as aspects, join points, pointcuts, and advice. These workflow languages therefore support a concern-based decomposition: the business logic which is the main concern in workflows can be specified in a modular way as a workflow process module and crosscutting concerns can be specified in a modularized manner using workflow aspects. Through such an approach, aspects can be used to obtain a cross-process view on how a concern is handled in various workflows. Thus if a programmer needs to understand or change a concern, he/she has to be concerned only about the relevant aspect.

In aspect-oriented progamming, aspects are weaved into the execution of a program using join points to provide alternative execution paths (Charfi and Mezini, 2007). For example, join points are well-defined points in the execution of the program. The behavioral code specified in the join point is known as *advice*. The advice code can be executed *before*, *after*, or *instead* of the join points. The work in (Charfi and Mezini, 2006) illustrates the application of aspect-oriented software development concepts to workflow languages to provide flexible

and adaptable workflows. AO4BPEL (Charfi and Mezini, 2007) is an aspect-oriented extension to BPEL that uses AspectJ to provide control flow adaptations (Kiczales *et al.*, 2001). Business rules can also be used to provide more flexibility during service composition. AO4BPEL enhances the limited capabilities of BPEL in terms of modularity and dynamic adaptability. XML files are used to provide functionality in AO4BPEL to avoid changing the service composition during runtime. In contrast with the standard WS-BPEL, AO4BPEL provides better support for functional modularization.

5 Failure Recovery Strategies for Web Services

In a service-oriented architecture, a business process can terminate successfully if all activities in it complete successfully or if the process is in a consistent state and the failed activities have been substituted by alternative execution paths. In practice, the great majority of business processes may encounter numerous and diverse failures. Failures can occur anywhere at any time due to the loosely-coupled, autonomous, and heterogeneous characteristics of the execution environment. An activity can fail in many ways, such as an undesirable return value, an unavailable resource, or even hardware failures. As a result, failure recovery for web services is an important issue.

As investigated in (Peltz, 2003), nearly 80% of process execution time is spent on handling exceptions. Considering parallel process execution, failure recovery and exception handling become considerably more difficult. One important reason is that a failed activity may have already affected another activity before recovery, as illustrated in Figure 1. In (Greenfield *et al.*, 2003), the authors use an e-procurement example to show that in many situations, compensation is not enough, pointing out that even if some aspects of an activity can be undone, it is not always the case that a process can return to its original state. Compensation of the failed activity therefore does not address the affect of the failure and recovery on dependent processes.

This section summarizes research on failure recovery techniques in web service composition. Most of these techniques build on strategies from advanced transaction models for compensation, retry, and contingency actions, but have been adapted for use in a service-oriented environment. Techniques for self-healing execution environments and checkpointing systems are also presented.

5.1 Re-do Strategy

One approach to keep a process running is to re-do (re-try) the failed activity. Re-do is the easiest way to handle a fault and keep running. However, sometimes re-do procedures are difficult to define in a business process.

Some fault-handling methods for job flow management were presented in (Tan *et al.*, 2010). The authors proposed a new business process execution model called BPEL4JOB. In this model, three fault-handling policies are designed. The cleanup policy gets the failure report and deletes the failed flow instance (assuming no side effect). The re-try policy uses a signal to indicate the job execution state and adds a while loop to each scope. The job will be executed repeatedly if the

signal indicates false. The third policy is the re-submission and instance migration method. This policy supports exporting job flow instance data in one flow engine, and importing it into another one so that the flow instance can resume. The challenge for this policy is to collect sufficient data from the source flow engine.

Another method for handling a re-do mechanism in BPEL was described by (Modafferi and Conforti, 2006). In this method, a re-do procedure is achieved by the event-handler and the compensation-handler. In the compensation-handler, both re-do and compensation procedures are defined following a select structure since only one compensation-handler can be defined for an activity. The aim of the event-handler is to set the variable that will drive the choice between redo and compensation.

In (Vaculín *et al.*, 2008), based on OWL-S, a recovery mechanism using semantic web services was introduced. In this recovery mechanism, a retry is used as a form of recovery action. A retry action can be defined in a fault-handler, or a constraint violation handler. The AP-Retry action defined in (Shrestha, 2010; Urban *et al.*, 2011d) also performs a re-execution of a portion of a process if a critical condition is violated in a process. The retry action in the assurance point model supports compensation back to a specified, logical recovery point in a process with the ability to recheck user-defined constraint conditions before the retry action.

5.2 Un-do Strategy

Once a crucial error occurs, it is important to clean all of the incorrect data that were generated by the failed activity. Typically, a process is recovered to a previous consistent state. The recovery procedure is usually done through the use of compensation.

In (Lakhal *et al.*, 2006), the authors used definition rules, composability rules and ordering rules to build a flexible web service composition model. In this model, for each compensatable activity, the users define a compensating procedure that will be invoked in case of a failure in the execution of an activity. The concept of vitality degree is also defined in this model. The vitality degree indicates that some activities are identified as optional, while others are tailored as crucial for the overall process.

A concept of automatic compensation was presented in (Wiesner *et al.*, 2008). Since the information of the effect about a service is defined as a process definition in OWL-S, it is possible to discover an automatic compensation based on the effect information. This technique is achieved by searching a service with effect ϵ^{-1} to undo the failed service which has the effect ϵ.

In a loosely-coupled execution environment which allows concurrent activity execution without isolation guarantee, the un-do strategy becomes more difficult to implement, because a failed activity may cause cascaded compensations. Dialani *et al.* (2002) proposed a transparent fault tolerance architecture for web services. The authors define a two layered model which consists of an application layer and a service layer. For failure recovery, the application layer implements two key components which are the global fault manager and the fault detector. The service layer includes a local fault manager which is a set of libraries that can be bound dynamically to the service code. In case of a failure, the local fault manager

tries to recover the fault first. In case a full recovery is not possible, the local fault manager recovers to a maximal state and escalates the fault notification to the global fault manager. Then the global fault manager initiates a roll back by notifying the affected services. The functionalities of the fault detector are sending the fault notification to the global fault manager and providing a dependency set for the current fault.

A fault handling method in decentralized web service composition was proposed by Chafle *et al.* (2005). The authors use a partition technique to decentralize web services. The decentralization algorithm partitions a scope in such a manner that the start and end of each scope reside in the same partition, which is referred to as the root partition of that scope. The fault handlers and compensation handlers are in the end of scope. In addition, each partition except the root has an inserted scope start and an inserted scope end which includes an inserted fault handler. When a fault occurs, the fault needs to be propagated to the root partition since the corresponding fault handler only resides in the root partition. Then the fault handler of the root partition of the scope in which the fault occurred, sends a DataCollection control message to its next partition(s) according to the control flow. The message flows along the path traversed by the fault (as per the fault propagation scheme). Each root partition except the top level scope for the composite service, enters a wait state. The fault handler and compensation handler in the top level root partition will then address the fault and compensate completed inner scopes respectively.

In the AP approach (Shrestha, 2010; Urban *et al.*, 2011d), if an error occurs or a critical condition is violated, the process will first be recovered back to a previous consistency point before invoking additional actions. The undo procedure supports shallow compensation and deep compensation. Shallow compensation provides the capability to execute one compensation action that reverses the affect of a composite group that has invoked multiple services. Deep compensation provides the capability to enter a completed composite group and execute individual compensating procedures for each service invoked by the composite group.

5.3 Alternative Strategy

Another approach used in failure recovery is to execute an alternative process, which was first introduced as a contingency in advanced transaction models. In many situations, alternative execution paths can totally substitute the failed activity so that the whole business can continue running.

5.3.1 Alternative with Un-do

An alternative method is a form of contingency. After failure recovery, the whole process backs up to a consistent state. However, in many cases, the re-do of the failed activity is still unsuccessful. Hence, an alternative execution path can be used to provide a means for maximizing forward recovery.

A replace operation was provided in (Vaculín *et al.*, 2008). A ReplaceBy(otherProcess) tries to use another process as a substitute for a failed process. In (Wiesner *et al.*, 2008), a more flexible operation named ReplaceByEquivalent was introduced. Because the OWL-S process model defines inputs, outputs, preconditions, and effects by using existing algorithms for

automatic web service discovery (matchmaking) (Sycara *et al.*, 2003), the information is used to dynamically find an alternative service. Based on ReplaceByEquivalent, the authors also give a definition of advanced backward and forward recovery. After a failure, a rollback is performed first for all processes that have finished at the same level. Then, if ReplaceByEquivalent can find an alternative service, it is executed. Otherwise, the backward recovery is repeated at one higher level in the hierarchy.

The AP-Cascaded Contingency action of the Assurance Point System (Shrestha, 2010; Urban *et al.*, 2011d) is a backward recovery process that searches backwards through the hierarchical nesting of composite groups to find a possible contingent procedure for a failed composite group. Once the contingent procedure is found, the process will perform forward recovery through execution of the contingent procedure.

5.3.2 Alternative without Un-do

In some failure recovery approaches, the failed activity is not crucial and alternative execution can be used to keep the business process running. Generally, each activity has several back-up activities in these approaches.

The Primary-Backup method (Zhang *et al.*, 2004) uses a backup service to substitute the failed primary service in grid services. In this model, each primary service has one or more backup services. Before replying to the client, the primary service needs to send the execution state to every backup service. If these backup services receive a failure notification, or do not receive a heartbeat message after a certain period of time, these backups need to cooperate to elect a new primary service. The newly elected primary service then sends a failover notification to the client so it can obtain a new server instance handle.

The merit of this approach is that it saves the expensive rollback or compensation of the failed activities. However, because the failed activity is just abandoned, the alternative strategy does not support the atomicity point of the ACID properties of the traditional transaction. Traditional transaction concepts require either all operations to complete or none to complete. Furthermore, an alternative without un-do method usually does not consider concurrent process execution errors. For example, if other concurrent activities have data dependencies on the failed activity, the results of these concurrent activities become questionable.

5.4 Other Techniques for Failure Recovery

As indicated in a previous section, the WS-BPEL standard provides a fault-handler, a compensation-handler, and a termination-handler attached to a scope to handle execution exception. To continue the process execution in case an exception occurs, the fault-handler might invoke the compensation-handler first to un-do the completed portion in the scope. If the corresponding compensation-handlers are not specified, then the default compensation-handler is assigned to the scope. However, in some cases, the default compensation-handler may cause complications and return unexpected results. Khalaf *et al.* (2009) highlight the two main problems with the fault and compensation mechanism in the current BPEL standard: 1) compensation order can violate control link dependencies

if control links cross the scope boundaries, and 2) high complexity of the default compensation order results due to the default handler behavior. Instead of the standard fault and compensation mechanism in BPEL, Khalaf *et al.* (2009) proposed a new and deterministic mechanism to better handle default compensation for scopes. In the new mechanism, the relationships between scopes include both structured nesting and graph-based links. Therefore, in case of an execution exception, the model can calculate the default compensation order before starting the compensation procedure.

Techniques are being devised to advance BPELs standard behavior in fault and exception handling and failure recovery. BPEL4Job (Tan *et al.*, 2010) is a BPEL-based advanced fault handling design for job flow management in a distributed environment. BPEL4JOB supports job flow modeling integrated with fault handling policies - cleanup, task level re-try and flow instance re-submit and, a set of fault handling schemes including a method for instance migration between flow engines in the distributed environment. A solution for automated and dynamic exception handling has been developed in (Christos *et al.*, 2007). The proposed framework, Service Relevance and Replacement Framework (SRRF), uses a pre-processor that enhances BPEL scenarios with code that detects failures, discovers alternate web services that can be used and invokes them, thereby resolving the exception.

Facilitating the automation of web service discovery, execution, composition and interoperation is becoming increasingly necessary (Yahyaoui *et al.*, 2010; Ukey *et al.*, 2010; Di Martino, 2009). Self-healing mechanisms are therefore being developed to monitor service compositions and, detect as well as recover from failures automatically (Chan *et al.*, 2009). Baresi *et al.* (2004) have presented Defensive Process Design and Service Run-time Monitoring as mechanisms for dynamic discovery of errors. The first approach designs a process such that it can recover from common faults that occur at run-time. Service run-time monitoring verifies if services are providing the functionalities they are meant to provide.

Ouyang *et al.* (2005) introduced WofBPEL, as a tool for automated analysis of BPEL processes. WofBPEL is an approach for static or off-line fault detection. This tool can analyze composite services after they are translated into the Petri Net Markup Language (PNML). The authors of Ouyang *et al.* (2005) also presented a companion tool, BPEL2PNML to translate BPEL process definitions into PNML. WofBPEL is then used to perform static analysis such as detection of unreachable actions and conflicting activities, on the output produced by BPEL2PNML.

An approach to specify monitoring directives and weave them dynamically into their corresponding process was presented in (Baresi and Guinea, 2005). BPELCheck (Fischer *et al.*, 2008) is a tool that has been developed to check BPEL processes for consistency violations by checking pre-defined conditions. Some examples of tools developed for monitoring and verification of web service compositions are ASTRO (Trainotti *et al.*, 2005), WSAT (Fu *et al.*, 2004) and LTSA-WS (Foster *et al.*, 2006). More such BPEL process verification techniques and tools have been discussed in (Van Breugel and Koshkina, 2006). A self-healing plug-in that has been proposed for use with WS-BPEL engines to enhance the ability of standard engines to provide process-based recovery actions is SH-BPEL (Modafferi *et al.*, 2006). The authors discuss the addition of some annotations

for enabling recovery actions, pre-processing based recovery mechanisms and, extended recovery mechanisms. The authors of (Modafferi and Conforti, 2006) proposed five mechanisms for enabling recovery actions in BPEL: external variable setting, timeout, redo, future alternative behavior, and rollback and conditional re-execution of the flow. They argue that enhancing BPELs capability for failure recovery through these mechanisms is a necessity to support emerging self-healing systems. In (Baresi *et al.*, 2007), Dynamo has been presented as a solution to self-healing BPEL compositions. Dynamo is an assertion-based framework that uses two special purpose languages: Web Service Constraint Language (WSCoL) to specify constraints such as pre- and post- conditions, and, Web Service Recovery Language (WSReL) to specify the recovery strategies.

Techniques for formalizing BPEL such as petri-net (Hamadi and Benatallah, 2003), pi-calculus (Sangiorgi and Walker, 2003), and model checking (Kovács *et al.*, 2007) guarantee consistency and monitor the execution to detect failures.

In checkpointing systems, consistent execution states are saved during the process flow. During failures and exceptions, the activity can be rolled back to the closest consistent checkpoint to move the execution to an alternative platform (Luo, 2000). The work in (Dialani *et al.*, 2002) uses the means of checkpointing and rollback to detect and recover the faults. The rule-based technique can be also used with checkpointing systems. Marzouk *et al.* (2009) introduced a periodic checkpointing based approach for strong mobility of orchestration processes. With a set of rules, WS-BPEL processes can be transformed to equivalent mobile ones. This approach can be used as a self-healing mechanism that supports resuming the execution of a failed process instance starting from the last checkpoint.

6 Conclusion and Future Work

This paper has presented current research that addresses transactional issues for data consistency and dynamic recovery in web service composition, illustrating the foundation that has been provided for this work through past research with advanced transaction models and transactional workflows. Standards provide coordination techniques that support the traditional 2PC approach as well as more relaxed techniques for managing long-running activities. From a research perspective, however, relaxed locking techniques and data dependency analysis are being investigated as a means to provide a more optimistic approach to supporting shared data access among concurrently executing processes. Furthermore, by adding modularization techniques, such as that of the assurance point system and concepts form aspect-oriented programming, user-defined correctness conditions, recovery techniques that combine backward and forward recovery actions, and support for addressing cross-cutting concerns have been investigated to provide more flexibility in execution and recovery options for service composition. Checkpointing and self-healing execution environments complement service composition and recovery techniques, providing autonomic ways to keep processes executing through mobility among different service platforms.

Due to the distributed nature of services, effective ways to deal with the transactional aspects of service composition and recovery is a challenging research topic. Even BPEL, the de-facto standard for composing web services, still lacks

sophistication with respect to handling faults, exceptions, and failure recovery. Future research will focus on more intelligent event and rule-driven techniques that support modularization in the specification of service composition, data constraints, and flexible recovery actions. More intelligent, de-centralized execution environments are also needed that understand the semantics of service composition and recovery techniques and apply self-healing approaches to monitor failures, detect data dependencies, and respond to failures and exceptional events. Such environments are needed to support the mobility of process execution when failures occur and to also address the impact that the failure and recovery of one process can have on other data dependent processes.

Acknowledgments

*This research has been supported by the National Science Foundation under Grant No. CCF-0820152. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

Baresi, L. and Guinea, S. (2005). Towards dynamic monitoring of ws-bpel processes. *Service-Oriented Computing-ICSOC 2005*, pp. 269–282.

Baresi, L., Ghezzi, C., and Guinea, S. (2004). Towards self-healing service compositions. In *Proceedings of First Conference on the Principles of Software Engineering*, volume 42, pp. 27–46.

Baresi, L., Guinea, S., and Pasquale, L. (2007). Self-healing bpel processes with dynamo and the jboss rule engine. In *International Workshop on Engineering of software services for pervasive environments, the 6th ESEC/FSE Joint Meeting*, pp. 11–20.

Bennett, B., Hahm, B., Leff, A., Mikalsen, T., Rasmus, K., Rayfield, J., and Rouvellou, I. (2000). A distributed object oriented framework to offer transactional support for long running business processes. In *Middleware 2000*, pp. 331–348.

Blake, L. (2005). *Design and implementation of delta-enabled grid services*. Master's thesis, Deptment of Computer Science and Engineering, Arizona State University.

Cabrera, F., Copeland, G., Freund, T., Klein, J., Langworthy, D., Orchard, D., Shewchuk, J., and Storey, T. (2002a). *Web services coordination (WS-Coordination)*. BEA, IBM, and Microsoft Web Service Specifications.

Cabrera, F., Copeland, G., Cox, B., Freund, T., Klein, J., Storey, T., and Thatte, S. (2002b). *Web services transaction (WS-transaction)*. BEA, IBM, and Microsoft Web Service Specifications.

Cabrera, L., Copeland, G., Feingold, M., Freund, R., Freund, T., Joyce, S., Klein, J., Langworthy, D., Little, M., Leymann, F., *et al.* (2005). *Web services business activity framework (ws-businessactivity)*. BEA, IBM, and Microsoft Web Service Specifications.

Chafle, G., Chandra, S., Kankar, P., and Mann, V. (2005). Handling faults in decentralized orchestration of composite web services. In *Service-Oriented Computing-ICSOC 2005*, pp. 410–423.

Chan, K., Bishop, J., Steyn, J., Baresi, L., and Guinea, S. (2009). A fault taxonomy for web service composition. In *Service-Oriented Computing-ICSOC 2007 Workshops*, pp. 363–375.

Charfi, A. and Mezini, M. (2006). Aspect-oriented workflow languages. In *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE*, pp. 183–200.

Charfi, A. and Mezini, M. (2007). Ao4bpel: An aspect-oriented extension to bpel. *World Wide Web*, **10**(3), 309–344.

Christos, K., Costas, V., and Panayiotis, G. (2007). Enhancing bpel scenarios with dynamic relevance-based exception handling. In *Web Services, 2007. ICWS 2007. IEEE International Conference on*, pp. 751–758. IEEE.

Cichocki, A. (1998). *Workflow and process automation: concepts and technology*. Kluwer Academic Pub.

Di Martino, B. (2009). Semantic web services discovery based on structural ontology matching. *International Journal of Web and Grid Services*, **5**(1), 46–65.

Dialani, V., Miles, S., Moreau, L., De Roure, D., and Luck, M. (2002). Transparent fault tolerance for web services based architectures. *Euro-Par 2002 Parallel Processing*, pp. 107–201.

Eder, J. and Liebhart, W. (1995). The workflow activity model WAMO. In *Proceedings of 3rd International Conference on Cooperative Information Systems, Vienna*, pp. 87–98.

Elmagarmid, A. (1992). *Database transaction models for advanced applications*. Morgan Kaufmann.

Elmagarmid, A., Leu, Y., Litwin, W., and Rusinkiewicz, M. (1990). A multidatabase transaction model for interbase. In *Proceedings of the 16th International Conference on Very Large Data Bases*, pp. 507–518.

Elmasri, R. and Navathe, S. (2011). *Fundamentals of database systems (6th Edition)*. Addison Wesley.

Fischer, J., Majumdar, R., and Sorrentino, F. (2008). The consistency of web conversations. In *Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering*, pp. 415–418.

Foster, H., Uchitel, S., Magee, J., and Kramer, J. (2006). Ltsa-ws: a tool for model-based verification of web service compositions and choreography. In *Proceedings of the 28th international conference on Software engineering*, pp. 771–774.

Fu, X., Bultan, T., and Su, J. (2004). Wsat: A tool for formal analysis of web services. In *Computer Aided Verification*, pp. 394–395.

Garcia-Molina, H. and Salem, K. (1987). Sagas. *ACM SIGMOD Record*, **16**(3), 249–259.

Greenfield, P., Fekete, A., Jang, J., and Kuo, D. (2003). Compensation is not enough [fault-handling and compensation mechanism]. In *Enterprise Distributed Object Computing Conference, 2003. Proceedings. Seventh IEEE International*, pp. 232–239.

Hamadi, R. and Benatallah, B. (2003). A petri net-based model for web service composition. In *Proceedings of the 14th Australasian database conference-Volume 17*, pp. 191–200.

Jang, J., Fekete, A., and Greenfield, P. (2007). Delivering Promises for Web Services Applications. In *IEEE International Conference on Web Services, Salt Lake City, Utah, USA*, pp. 599–606.

Jordan, D., Evdemon, J., Alves, A., Arkin, A., Askary, S., Barreto, C., Bloch, B., Curbera, F., Ford, M., Goland, Y., *et al.* (2007). *Web services business process execution language version 2.0*. OASIS Standard.

Karnath, M. and Ramamritham, K. (1998). Failure handling and coordinated execution of concurrent workflows. In *Proceedings of 14th International Conference on Data Engineering*, pp. 334–341.

Khalaf, R., Roller, D., and Leymann, F. (2009). Revisiting the behavior of Fault and Compensation handlers in WS-BPEL. In *On the Move to Meaningful Internet Systems: OTM 2009*, pp. 286–303.

Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., and Griswold, W. (2001). An overview of AspectJ. In *Object-Oriented Programming, ECOOP 2001*, pp. 327–354.

Kovács, M., Varró, D., and Gönczy, L. (2007). Formal modeling of bpel workflows including fault and compensation handling. In *Proceedings of the Workshop on Engineering Fault Tolerant Systems*, pp. 1–es.

Kuo, D., Fekete, A., Greenfield, P., and Jang, J. (2002). Towards a framework for capturing transactional requirements of real workflows. In *Second International Workshop on Cooperative Internet Computing 2002*, pp. 113–122.

Lakhal, N., Kobayashi, T., and Yokota, H. (2006). Dependability and flexibility centered approach for composite web services modeling. *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE*, pp. 163–182.

Limthanmaphon, B. and Zhang, Y. (2004). Web service composition transaction management. In *Proceedings of the 15th Australasian Database Conference-Volume 27*, pp. 171–179. Australian Computer Society, Inc.

Liu, Z. (2009). *Decentralized data dependency analysis for concurrent process execution*. Master's thesis, Texas Tech University.

Luo, Z. (2000). Checkpointing for workflow recovery. In *Proceedings of the 38th Annual on Southeast Regional Conference*, pp. 79–80.

Marzouk, S., Maalej, A., Rodriguez, I., and Jmaiel, M. (2009). Periodic checkpointing for strong mobility of orchestrated web services. In *2009 Congress on Services-I*, pp. 203–210. IEEE.

Mikalsen, T., Tai, S., and Rouvellou, I. (2002). Transactional attitudes: Reliable composition of autonomous Web services. In *Workshop on Dependable Middleware-based Systems, the International Conference on Dependable Systems and Networks (DSN 2002)*.

Modafferi, S. and Conforti, E. (2006). Methods for enabling recovery actions in ws-bpel. *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE*, pp. 219–236.

Modafferi, S., Mussi, E., and Pernici, B. (2006). Sh-bpel: a self-healing plug-in for ws-bpel engines. In *Proceedings of the 1st Workshop on Middleware for Service Oriented Computing (MW4SOC 2006)*, pp. 48–53.

Moss, J. (1985). *Nested transactions: an approach to reliable distributed computing*. MIT. Press.

Moss, J. (1987). Log-based recovery for nested transactions. In *In Proceeding of 13th International Conference on Very Large Data Bases*, pp. 427–432.

Ouyang, C., Verbeek, E., van der Aalst, W., Breutel, S., Dumas, M., and ter Hofstede, A. (2005). Wofbpel: A tool for automated analysis of bpel processes. *Service-Oriented Computing-ICSOC 2005*, pp. 484–489.

Papazoglou, M. and Georgakopoulos, D. (2003). Service-oriented computing. *Communications of the ACM*, **46**(10), 25–28.

Peltz, C. (2003). Web services orchestration and choreography. *Computer*, pp. 46–52.

Ramachandran, J. (2011). *Integrating exception handling and data dependency analysis through application exception rules*. Master's thesis, Texas Tech University.

Riegen, M., Husemann, M., Fink, S., and Ritter, N. (2010). Rule-based coordination of distributed web service transactions. *IEEE Transactions on Services Computing (January-March 2010)*, **3**(1), 60–71.

Rolf, A., Klas, W., and Veijalainen, J. (1997). *Transaction management support for cooperative applications*. Kluwer Academic Pub.

Sangiorgi, D. and Walker, D. (2003). *The pi-calculus: a Theory of Mobile Processes*. Cambridge University Press.

Shrestha, R. (2010). *Using assurance points and integration rules for recovery in service composition*. Master's thesis, Texas Tech University.

Singh, M. and Huhns, M. (2005). *Service-oriented computing: semantics, processes, agents*. John Wiley & Sons Inc.

Sycara, K., Paolucci, M., Ankolekar, A., and Srinivasan, N. (2003). Automated discovery, interaction and composition of semantic web services. *Journal of Web Semantics*, **1**(1), 27–46.

Tan, W., Fong, L., and Bobroff, N. (2010). Bpel4job: a fault-handling design for job flow management. *Service-Oriented Computing–ICSOC 2007*, pp. 27–42.

Taniar, D. and Goel, S. (2007). Concurrency control issues in grid databases. *Future Generation Computer Systems*, **23**(1), 154–162.

Taniar, D., Leung, C., Rahayu, W., and Goel, S. (2008). *High performance parallel database processing and grid databases*, volume 67. Wiley.

Trainotti, M., Pistore, M., Calabrese, G., Zacco, G., Lucchese, G., Barbon, F., Bertoli, P., and Traverso, P. (2005). Astro: Supporting composition and execution of web services. *Service-Oriented Computing-ICSOC 2005*, pp. 495–501.

Ukey, N., Niyogi, R., Milani, A., and Singh, K. (2010). A bidirectional heuristic search technique for web service composition. In *Computational Science and Its Applications–ICCSA 2010, LNCS 6019*, pp. 309–320.

Urban, S., Liu, Z., and Gao, L. (2009a). Decentralized data dependency analysis for concurrent process execution. In *Enterprise Distributed Object Computing Conference Workshops, 2009. EDOCW 2009. 13th*, pp. 74–83.

Urban, S., Xiao, Y., Blake, L., and Dietrich, S. (2009b). Monitoring data dependencies in concurrent process execution through delta-enabled grid services. *International Journal of Web and Grid Services*, **5**(1), 85–106.

Urban, S., Gao, L., Shrestha, R., and Courter, A. (2010). Achieving Recovery in Service Composition with Assurance Points and Integration Rules. *On the Move to Meaningful Internet Systems: OTM 2010*, pp. 428–437.

Urban, S., Liu, Z., and Gao, L. (2011a). Decentralized communication for data dependency analysis among process execution agents and integration rules. to appear in *Volume 8, Issue 4 (October-December 2011) of the International Journal of Web Services Research*.

Urban, S., Gao, L., Shrestha, R., and Courter, A. (2011b). The dynamics of process modeling: new directions for the use of events and rules in service-oriented computing. In *The Evolution of Conceptual Modeling, LNCS 6520*, pp. 205–224. Springer.

Urban, S., Courter, A., Gao, L., and Shuman, M. (2011c). Supporting Data Consistency in Concurrent Process Execution With Assurance Points and Invariants. In *Rule-Based Modeling and Computing on the Semantic Web, RuleML 2011, America, LNCS 7018*.

Urban, S., Gao, L., Shrestha, R., Xiao, Y., Friedman, Z., and Rodriguez, J. (2011d). The Assurance Point Model for Consistency and Recovery in Service Composition. In *Innovations, Standards and Practices of Web Services: Emerging Research Topics, IGI Global publication*, pp. 250–287.

Vaculín, R., Wiesner, K., and Sycara, K. (2008). Exception handling and recovery of semantic web services. In *Fourth International Conference on Networking and Services, ICNS 2008*, pp. 217–222.

Van Breugel, F. and Koshkina, M. (2006). Models and verification of bpel. *Unpublished Draft*.

Wächter, H. and Reuter, A. (1991). *The contract model*. Universität, Fakultät Informatik.

Weikum, G. (1991). Principles and realization strategies of multilevel transaction management. *ACM Transactions on Database Systems (TODS)*, **16**(1), 132–180.

Wiesner, K., Vaculín, R., Kollingbaum, M., and Sycara, K. (2008). Recovery mechanisms for semantic web services. In *Distributed Applications and Interoperable Systems*, pp. 100–105. Springer.

Wodtke, D., Weißenfels, J., Weikum, G., and Dittrich, A. (1996). The Mentor project: Steps towards enterprise-wide workflow management. In *Proceedings of the Twelfth International Conference on Data Engineering*, pp. 556–565.

Worah, D. and Sheth, A. (1997). Transactions in transactional workflows. In *Transactions in Transactional Workflows*, pp. 3–34.

Xiao, Y. (2006). *Using deltas to analyze data dependencies and semantic correctness in the recovery of concurrent process execution*. Ph.D. dissertation, Arizona State University.

Xiao, Y. and Urban, S. (2008a). Process dependencies and process interference rules for analyzing the impact of failure in a service composition environment. *Journal of Information Science and Technology*, **5**(2), 21–45.

Xiao, Y. and Urban, S. (2008b). Using Data Dependencies to Support the Recovery of Concurrent Processes in a Service Composition Environment. In *Proceedings of the Cooperative Information Systems Conference (COOPIS), Monterrey, Mexico*, pp. 139–156.

Xiao, Y. and Urban, S. (2009). The DeltaGrid Service Composition and Recovery Model. *International Journal of Web Services Research*, **6**(3), 35–66.

Xiao, Y. and Urban, S. (2012). Using rules and data dependencies for the recovery of concurrent process in a service-oriented environment. to appear in *IEEE Transactions on Service Computing*.

Xiao, Y., Urban, S., and Dietrich, S. (2006). A process history capture system for analysis of data dependencies in concurrent process execution. *Data Engineering Issues in E-Commerce and Services*, pp. 152–166.

Yahyaoui, H., Maamar, Z., and Boukadi, K. (2010). A framework to coordinate web services in composition scenarios. *International Journal of Web and Grid Services*, **6**(2), 95–123.

Zhang, X., Zagorodnov, D., Hiltunen, M., Marzullo, K., and Schlichting, R. (2004). Fault-tolerant grid services using primary-backup: feasibility and performance. In *IEEE International Conference on Cluster Computing, 2004*, pp. 105–114.

Zhao, W., Moser, L., and Melliar-Smith, P. (2005). A reservation-based coordination protocol for Web Services. In *IEEE International Conference on Web Services, Orlando, Florida , USA*, pp. 49–56.