MASTER OF SCIENCE THESIS PROPOSAL

DEPARTMENT OF COMPUTER SCIENCE

TEXAS TECH UNIVERSITY

INTEGRATING EXCEPTION HANDLING AND DATA DEPENDENCY ANALYSIS THROUGH APPLICATION EXCEPTION RULES*

by

Janani Ramachandran

Committee Members:

Dr. Susan Urban (Chair)

Dr. Susan Mengel

November 2010

*This research is supported by NSF Grant No. CCF-0820152.

1. Introduction

Past years have witnessed the growth of the Internet from a medium for simple communication into a powerful platform that supports evolving online business enterprises. This has led to the increased usage of Web Services and Service-Oriented Computing (SOC) to enhance and support online business transactions for both business-to-business (B2B) and business-to-consumer (B2C) applications. In service-oriented computing, services are used as the fundamental units to develop solutions. The application of SOC on the web is realized by the use of Web Services (Papazoglou and Georgakopoulos 2003).

In traditional distributed computing environments, data consistency among distributed transactions is guaranteed by the application of the two-phase commit protocol to support the ACID properties of atomicity, consistency, isolation and durability. Also, serializability is ensured by the use of locking protocols (Elmasri and Navathe 2010). However, in a service-oriented environment, processes that are composed of services cannot be treated like traditional transactions that satisfy ACID properties since the individual services are loosely-coupled and autonomous in nature. Moreover, it is not feasible to enable services to lock data for the duration of the process they constitute since these processes may be long running. Therefore, in such an environment, it is necessary to relax the isolation requirement. Due to this, the correctness of concurrently running processes that access shared data might be affected. Thus, ensuring the semantically correct execution of concurrent transactions by maintaining data consistency in a service-oriented environment is an important issue.

Data consistency issues for concurrently running processes has been the major focus of the Decentralized Data Dependency (D3) Analysis Project. The primary objective of the D3 project has been to develop a decentralized approach to data dependency analysis and failure recovery among concurrently executing processes. As part of achieving this objective, an approach to identify dependencies among concurrent processes that may potentially lead to inconsistencies was proposed in (Xiao 2006). Based on this approach, algorithms to determine data dependencies in a decentralized manner were developed in (Liu 2009). These concepts addressed the need to maintain consistency by identifying processes that may be affected by the recovery of a failed process but lacked support to detect and handle inconsistencies dynamically. A more flexible service composition and recovery model was developed in (Shrestha 2010) through the use of Assurance Points in association with events and rules to provide a flexible way of checking constraints and responding to failures. Shrestha's work was developed as an extension to past work on the service composition and recovery model in (Xiao and Urban 2009) to support approaches for flexible failure recovery.

Ideally, the work of (Liu 2009) and (Shrestha 2010) needs to be integrated so that the partial recovery of a process can also use data dependency analysis to identify processes that may be data dependent on the recovered process. The recovery techniques proposed in (Shrestha 2010) can be used to dynamically respond to internal events while checking for consistency, where internal events are errors that occur during process execution. But to achieve efficiency in exception handling, the process execution environment also needs enhanced capabilities to have variability of response to external events based on process execution status, where external events are interrupts received from the environment external to a process. Furthermore, the work in (Shrestha 2010) addresses the flexible recovery of a particular process that has failed. But in a concurrent process execution environment, it is also important to address the

consistency issues that may arise due to data dependencies between concurrently executing processes. That is, failure recovery of a process needs to be communicated to the dependent processes so that they may also recover if required. Therefore, processes in concurrent execution need to have flexible constraint checking and recovery capability, handle exceptions efficiently, and maintain consistency in the environment by communicating failure recovery to dependent processes.

This proposal presents the use of Application Exception Rules, together with the use of Data Dependency Analysis, to provide an efficient and flexible way of responding to exceptions, and maintaining data consistency in concurrent process execution. Similar to Integration Rules (Jin 2004; Urban et al., 2001), Application Exception Rules are triggered by the occurrence of events. An Application Exception Rule (AER) is a rule that specifies recovery actions to be carried out based on the execution status of the corresponding process. The concept of AERs provides a flexible way of using rules to respond to internal and external events. Using these rules will facilitate processes to handle exceptions depending on their execution status, instead of giving a fixed response to all exceptions. During process execution, when an exception occurs, the exception is communicated to the process as an event. This event interrupts process execution and triggers the corresponding Application Exception Rule, which checks the process execution status based on the critical points that have been crossed during process execution. These critical points are determined using the Assurance Point (AP) concept (Shrestha 2010). Based on the most recently crossed AP, the AER fires the corresponding recovery action. This requires AERs to work in association with APs and the recovery techniques described in the AP model. Therefore, providing the functionality of AERs will involve extending the AP model. This concept, combined with the Data Dependency Analysis capability will provide an approach to flexible exception handling along with maintaining data consistency.

The remainder of this proposal is structured as follows. Section 2 gives an overview of related work. Background on Decentralized Data Dependency Analysis and Assurance Points is given in Section 3. Section 4 presents the statement of objectives for the proposed research, followed by a discussion of the objectives in Section 5. Section 6 presents the proposed timeline for the research. A brief summary of the proposal is given in Section 7.

2. Related Work

A fault is defined as an abnormal condition at the component or sub-system level and may lead to a failure. Faults oscillate between active and dormant states. A web service fails when a fault becomes active (Chan et al., 2007). An exception is a situation that is raised to indicate the occurrence of errors, faults or failures. It is important to build exception aware systems so that they can learn, correct themselves and evolve (Luo et al., 2000). According to (Brambilla et al., 2005), the exceptions for a process can be classified into three categories: behavioral or user-generated exceptions caused by incorrect execution order of process activities; semantic or application exceptions caused by unsuccessful logical outcomes of process activities execution; and system exceptions caused by the malfunctioning of the application infrastructure. Failures lead to deviation and prevent business processes from completing normally and therefore need to be handled to ensure system consistency (Greenfield et al., 2003).

Some techniques that have been used to handle failures are rollback, forward recovery through contingency, backward recovery through compensation, retry, and even manual intervention (Eder and Liebhart 1996). The work in (Greenfield et al., 2003) argues that such standard approaches of fault handling and compensation mechanisms are not enough to manage the fault tolerance and failure recovery in workflow systems. Also, services are more prone to failures in a service composition compared to atomic services because of the dependency of composite services on external services. In a service composition, failures may be caused by service unavailability, concurrency, dependency, inconsistency, or incorrect composition. Thus, stronger mechanisms are required to handle failures, address dynamic failure recovery and build more fault tolerant systems. The following sections outline some of the related work in these areas.

2.1 Fault Handling in BPEL

Techniques are being devised to advance the standard behavior of the Business Process Execution Language (BPEL) in fault and exception handling and failure recovery. BPEL4Job (Tan, Fong, and Bobroff 2007) 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, Costas, and Panayiotis 2007). The proposed framework known as 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.

An approach to specify monitoring directives and weave them dynamically into their corresponding process is presented in (Baresi and Guinea 2005). BPELCheck (Fischer, Majumdar, and Sorrentino 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, Bultan, and Su 2004) and LTSA-WS (Foster et al., 2006). More such BPEL process verification techniques and tools have been discussed in (Breugel and Koshkina 2006).

Several projects in recent years have focused on using agents in business transaction models. Agents are self-contained, autonomous entities that are capable of achieving tasks through observation and learning. Huhns (Huhns 2002) compares agents and web services and points out several ways in which agents extend web services. The use of agent-based web services is a current topic of research. ADEPT (Jennings et al., 2000), an agent-based design and implementation approach, implements a robust and flexible business process management system in which the agent-based approach enables the businesses to retain the autonomy of the transactions. The work in (Müller, Grener, and Rahm 2004) presents AGENTWORK as a workflow management system that supports dynamic and automated workflow adaptations. In this system, a rule-based approach is followed to specify exceptions and the required workflow adaptations. The authors of (Chakravarty and Singh 2007) propose an event-driven approach for agent-based business process management to support exception handling. The work in (Shen, Ghenniwa, and Li 2006) describes an agent-based service-oriented system architecture to facilitate automated and dynamic collaboration among businesses. In this system, web services are implemented by

the Agent-based Web Services (AWS) technology which considers the web service environment as a collection of software agents implemented as services with different capabilities and functionalities.

With research involving agents in the service environment, the concept of self-healing has also been gaining popularity. 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, Mussi, and Pernici 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) propose 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 BPEL's capability for failure recovery through these mechanisms is a necessity to support emerging self-healing systems.

2.2 Aspect-Oriented Programming

Aspect-Oriented Programming (AOP) (Kiczales et al., 1997) has been proposed as a technique for modularization and improving the separation of concerns in software. In AOP, three key concepts are used: join points, pointcuts and advice. Join points are checkpoints in the execution of a program such as method calls, constructor calls. A pointcut is used to determine related join points. The advice specifies the crosscutting functionality at join points identified by a pointcut. The advice code is executed when a join point identified by a pointcut is reached. This code may be executed before, after, or instead of the join point currently reached. An aspect module consists of several pointcut definitions and their corresponding advice code. In AOP, aspects are woven into the program execution.

Current workflow languages do not support modularization of concerns that span across process boundaries such as data validation and security. Due to this, the code of the crosscutting concerns is spread across several workflows and is tangled with the code addressing other concerns. This leads to complex workflow process specifications that are difficult to comprehend, maintain, modify, and reuse. To address these issues, (Charfi and Mezini 2006) proposes the use of aspect-orientation concepts in workflow languages. Aspect-oriented workflow languages are introduced which 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 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. This concern-based decomposition of workflow process specification by the use of aspect-oriented workflow languages has been validated by the development of AO4BPEL (Charfi and Mezini 2007), which is an aspect-oriented workflow language for web service composition. It is an aspect-oriented extension to BPEL and provides support for modularization of crosscutting concerns and dynamic changes in BPEL.

2.3 Checkpoints, Events and Rules

The technique of using checkpointing for failure recovery and exception handling has been discussed in (Luo 2000). In this technique, consistent states in transaction execution are saved as checkpoints. When a

failure occurs, workflows restore a checkpoint by rolling back and resuming their execution from that checkpoint. Events and rules can also be used with business processes to enforce constraints, detect failures and define appropriate recovery mechanisms. An event is the occurrence of an activity that may signify a problem, an opportunity or a deviation (Michelson 2006). Other than controlling the flow of business process execution, events and rules can be used as a mechanism for handling failures and exceptions. Event Condition Action (ECA) rules can be used to specify actions that need to be automatically carried out on the occurrence of an event. In (Liu et al., 2007), ECA rules have been used for handling and recovering from faults. These rules are integrated with business logic to generate fault-tolerant BPEL processes in order to overcome BPEL's limited fault handling capability. The work in (Luo et al., 2000) proposes the use of "Justified" ECA to allow the use of context-based reasoning to enhance the exception handling in business processes.

The work in (Shrestha 2010) proposes an approach that builds on the concept of checkpointing and enhances it with the use of events and rules. It presents the use of Assurance Points, combined with the use of events and rules, to provide a flexible way of checking for consistency and responding to internal events. APs store critical data related to process execution and can serve as checkpoints. During process execution, APs check pre-, post- and application conditions by invoking corresponding integration rules. This enables the check for consistency of the process with respect to user defined constraints. When failures occur, APs are used as rollback points that recheck preconditions and determine whether to invoke forward or backward recovery actions.

2.4 Data Consistency Techniques for Services

This section discusses some of the techniques that have been proposed for ensuring consistency in a service composition.

2.4.1 Promises

One of the issues in the design of complex web-service based applications is the lack of support for isolation. Execution of concurrent applications in such an environment may lead to semantically incorrect conditions, dirty reads and lost updates (Jang, Fekete, and Greenfield 2007). The promises approach was proposed to address this issue by providing the isolation support. The goal of this approach is to enable the applications in an SOC environment to have the benefits of isolation.

A 'promise' is a form of agreement between a service i.e., a promise maker, and a client application i.e., a promise client. When a service grants a promise to a client, it 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. In the model discussed in (Greenfield et al., 2007), a Promise Manager works with resource managers and application services to check resource availability, grant or deny promises, and make sure that the granted promises are not violated. Client applications determine the constraints they need to have over a set of resources, express them as predicates, and send them to the relevant promise manager within a promise request. Every promise request contains a set of predicates, a set of resources and, a promise duration. The promise manager will then check the existing set of promises, check for the availability of resources by corresponding with the resource managers, and either grant or reject the promise request by sending back a promise response. When a promise request is accepted, a promise duration is fixed, indicating how long the promise manager guarantees to keep its promise to the client. This might be the

same as or lesser than the duration included in the promise request of that client. Once a promise is granted to a client application, it gets isolated from the effects of concurrent execution activities on the resources protected by that promise. Such a client can make changes to resources protected by its promises successfully if such changes do not violate the constraints implied by its promises. The clients release their promises by sending promise release messages to their promise managers. If a promise request expires i.e., exceeds the promise duration fixed in the promise response, the corresponding promise manager returns a 'promiseexpired' error to the client if it tries to perform operations under the protection of the expired promise.

One of the general mechanisms to implement promises is 'soft locks'. This method shows whether an item is allocated or reserved for clients by recording this information as a field in the database records. This does not lock the records of allocated or reserved items from being accessed. Instead, this field is read when checking for available resources and corresponding records are simply ignored if the field indicates that the item is allocated. Traditional locking mechanisms to ensure isolation can be seen as a very strong form of promise. The promises approach is weaker but is effective as it is more precise. The main advantage of this approach is that it allows the client applications to be very specific with respect to conditions they need held over resources, thereby allowing other promises over the same resources to be granted simultaneously as long as they do not conflict with existing promises. Once promises are granted, clients can continue processing without the effects of concurrency in the execution environment.

2.4.2 Tentative Hold

The Tentative Hold technique (Limthanmaphon and Zhang 2004) provides another 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 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. It allows a resource owner to query existing holds granted by it, cancel existing holds granted by it, and query activities. The resource coordinator is also responsible to notify affected client coordinators when a relevant resource becomes unavailable. Thus this approach provides a non-blocking mechanism of temporarily holding resources.

2.4.3 Reservation-Based Coordination

The reservation based approach proposed in (Zhao, Moser, and Melliar-Smith 2005) serves the purpose of temporarily locking data, similar to the promises and tentative hold techniques. This technique 'reserves' resources for use based on resource requests. In this protocol, each task within a business

activity is executed in two steps. Each step is executed as a single traditional transaction. The first step reserves resources according to business logic. This reservation forms a contract between the client and the resource provider. A 'fee' is associated with the reservation proportional to the duration of the reservation, to discourage applications from reserving the resources for extended periods of time. The second step involves confirming or rejecting the reservation in accordance with the business rules. Since the first step involves explicit reservation of the resources, the application can choose to either continue with the process or go backward, as the reservation may have either effect. After the first step of reservation by an application, other transactions cannot make any assumptions about the future of the resources reserved by that application. 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 done 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. Once a participant has accepted a reservation, it is committed to the reserved resource until the coordinator cancels the reservation. In traditional transactions, any of the participants might affect the transactions to rollback or abort. But in the reservation-based coordination protocol, only the client is authorized to commit or rollback the business transaction through its coordinator.

One of the advantages of this technique over traditional locking is that if a resource is reserved and a different transaction wants to access it, the transaction can be informed that the resource is already reserved. So instead of waiting as in traditional locking, the transaction can take necessary and appropriate actions to proceed further without waiting for the resource to be available. Therefore, the reservation-based approach provides a way to avoid traditional locking while maintaining the ACID properties.

2.4.4 Transactional Attitudes

The work in (Mikalsen, Tai, and Rouvellou 2002) proposes the Web Service Transaction (WSTx) framework to address the issue of transactional reliability in web service composition. The WSTx framework introduces transactional attitudes that enable web service providers to explicitly describe their specific transactional semantics and capabilities, and web service clients to describe their transactional requirements. The transactional attitudes are used to communicate otherwise implicit transactional semantics and requirements while maintaining the autonomy of the individual transactional web services. This enables a clear separation of the participants' transactional properties from other aspects of a service description.

The authors of (Mikalsen, Tai, and Rouvellou 2002) define transactional attitudes for the web service providers as Provider Transactional Attitudes (PTAs) and the transactional attitudes for clients as Client Transactional Attitudes (CTAs). PTAs allow the web service providers to explicitly describe their transactional semantics and are therefore useful in automating the composition of web services into larger transactional patterns. CTAs allow the clients to describe their expectations explicitly. Each client executes one or more actions within the scope of a web transaction, where each action is a provider transaction executing within the scope of a larger web transaction. The transactional attitudes can therefore be used to automate reliable execution of individual web transactions. In the WSTx framework, such automation is supported by the middleware system. The middleware acts as an intermediary between a client and multiple web service providers. It maps the client transactional requirements (associated with

CTAs) with the various transactional capabilities and semantics of the web service providers (associated with PTAs). This helps in a reliable execution of web transactions through the use of transactional attitude descriptions.

2.4.5 Decentralized Data Dependency Analysis for Concurrent Process Execution

The work in (Liu 2009) provides techniques to perform data dependency analysis in a concurrent process execution scenario. The central focus in (Liu 2009) is to identify concurrently executing processes that are write-dependent and potentially read-dependent on a failed process. The idea here is to ensure data consistency by recovering processes that are dependent on a failed process, in addition to recovering the failed process itself. The concept in (Liu 2009) builds on the approaches discussed in the previous subsections and locks data items accessed by the failed process and the identified dependent processes temporarily, till all of these processes are recovered. Therefore, in the execution environment, only the concurrently running processes that need access to the locked data items need to wait temporarily. The other processes continue running uninterrupted.

3. Background

To provide the appropriate background for the proposed research, this section discusses the work on decentralized data dependency analysis, and assurance points in the context of the DeltaGrid service composition and recovery model from (Xiao, and Urban 2009). Section 3.1 will first summarize the data dependency analysis work of (Liu 2009). Section 3.2 will then summarize the concept of Assurance Points from (Shrestha 2010). Section 4 will then present objectives that represent an extension and integration of this research.

3.1 Decentralized Data Dependency Analysis

As described in (Urban et al., 2009), then a Process History Capture System (PHCS) merges deltas from distributed service executions to create a centralized, time-sequenced global delta object schedule. This object schedule is then used to analyze data dependencies between various process executions and determine how the failure and recovery of one process can potentially affect other processes. The work in (Urban et al., 2009) demonstrates this model and discusses the overhead associated with the centralized approach to data dependency analysis. The work in (Liu 2009) developed a decentralized approach to analyzing data dependencies in a service composition. Liu's work developed the concept of Process Execution Agents (PEXAs), which are capable of capturing deltas locally. Each PEXA communicates with other PEXAs to determine data dependencies between distributed service executions. The internal architecture of a PEXA is described in detail in (Liu 2009).

In the decentralized data dependency analysis approach, PEXAs are responsible for the execution of local processes. Each PEXA also maintains a local delta object schedule. This enables the determination of data dependencies between the locally executing services. Since a process can execute services at multiple sites, with each site being monitored by a different PEXA, every PEXA must communicate with other PEXAS to determine a global view of the process data dependencies. This global view is provided by the construction of a graph known as the Process Dependency Graph.

The work in (Liu 2009) also developed two different algorithms for decentralized data dependency analysis - the Lazy Approach, and the Eager Approach. The algorithm used in the Lazy approach assumes

that every process runs successfully and starts building the process dependency graph only when one of the processes fails due to the failure of one of its services. The algorithm used in the Eager approach builds the dependency graph dynamically, at runtime. This makes sure that a global process dependency graph is available as soon as a process fails.

One of the limitations of the work in (Liu 2009) is that the data dependency analysis procedure is not fully integrated into a flexible service composition model. In (Liu 2009), when a process fails, the current algorithms assume that the failed process and all dependent processes recover by rolling back completely to the beginning of their execution. Therefore the approaches discussed perform the data dependency analysis at the process level. The proposed work will investigate how to integrate the data dependency analysis process into a service composition and recovery model enhanced with APs for more flexible recovery actions.

3.2 Assurance Points

According to the service composition and recovery model in (Xiao, and Urban 2009), a process is a top level execution entity hierarchically composed of different types of execution entities. A process is denoted as p_i, where p represents the process and the subscript i represents the unique identifier of the process. An operation represents a service invocation. An operation in process p_i is denoted as $op_{i,j}$, where op is the operation, i represents process p_i and j represents the unique identifier of the operation within the process p_i. Compensation, denoted as $cop_{i,j}$ is an operation for backward recovery and contingency denoted as $top_{i,j}$ is an operation for forward recovery. In this model, service execution failure recovery is made flexible by adding scopes within the context of a process execution. Atomic groups and composite groups are logical execution entities that enable the specification of the complex control structure of a process. An atomic group is an execution unit that contains an operation, an optional compensation, and an optional contingency. A composite group is an execution unit that may be composed of multiple atomic groups and/or multiple composite groups that execute in sequence or parallel, an optional compensation, and an optional contingency. The entire process can be considered to be the top level composite group. Contingency is an operation that provides an alternate execution for forward recovery. Upon the failure of a group, contingency is tried first. If contingency fails or is not available, compensation is invoked. Compensation is the backward recovery operation that logically undoes the outcomes of completed activities. While execution of the compensating procedure attached to a composite group is known as shallow compensation, deep compensation involves individually compensating each entity contained in the composite group.

Although the model in (Xiao, and Urban 2009) uses rules for process recovery during failures, it does not provide approaches for flexible constraint checking and dynamic response to events. To address this, (Shrestha 2010) extends the model with the concept of Assurance Points. An AP has been defined as a process execution correctness guard and a potential rollback point during forward and backward recovery activities. APs are inserted at critical points in a process to check for consistency and reduce the risk of failure. They can also be used as checkpoints for recovery procedures. They are used together with integration rules to support correctness and consistency checking.

An AP is defined as AP = <apId, apParameters, IR_{pre}, IR_{post}, IR_{cond}> where, apID uniquely identifies the AP, apParameters is the list of critical data items stored in the AP, IR_{pre} is an integration rule that defines

a pre-condition, IR_{post} is an integration rule that defines a post-condition and, IR_{cond} is an integration rule that defines additional application rules. There may be 0 or more occurences of apParameters and IR_{cond} and 0 or 1 occurrence of IR_{pre} and IR_{post}. The IRs that are part of APs are expressed as ECA rules as shown in Figure 1. When process execution reaches a specific AP, the IRs corresponding to it are trigerred. When an IR is invoked, its condition is checked. If the condition evaluates to true, the action specified is carried out. For IR_{pre} and IR_{post}, the condition is always expressed as the negative form of the constraint so that the corresponding action is executed whenever the constraint is not satisfied. Thus, the action part of these IRs can be used to specify recovery activities which will be performed on the violation of specific constraints. This way, APs serve as consistency checks and support recovery to handle any inconsistencies.

On reaching a particular AP, the condition associated with IRpost is evaluated first to validate the execution of the previous entity in process execution. If the post-condition is violated, the corresponding recovery action is carried out. If the post-condition is satisfied or there is no IR_{post} in the AP, the condition of IR_{pre} is checked. If the pre-condition is violated, the corresponding recovery action is carried out. If the pre-condition is satisfied or there is no IR_{re} in the AP, any IR_{cond} that may have been specified are checked. The IR_{cond} rules invoke additional parallel activity based on the application conditions and do not alter the normal flow of process execution.

The most basic recovery action is to invoke an alternative process. The recovery actions associated with APs can also be one of the following.

- APRollback: This recovery action is used when the entire process needs to be rolled back through compensation, to the start of the process.
- APRetry: This is used when the process needs to be compensated to a specific AP. The APRetry procedure can optionally specify the particular AP to which the process needs to be backward recovered. By default, APRetry compensates the process to the first AP reached through compensation, within the same scope. When the specified/first AP is reached, the pre-condition defined at that AP is first re-checked. If the condition is violated, the corresponding action is executed. Otherwise, process execution is resumed from that point to re-execute compensated activities. In such a case, there is a possibility that the process may encounter the same inconsistency that caused APRetry to be invoked. When this happens, action 2 (see Figure 1) is invoked instead of action 1.
- APCascadedContingency (APCC): This recovery action searches backwards through the hierarchical nested structure of the process for a possible contingent procedure for a failed execution entity in the process. During the APCC backward recovery process, when an AP is reached, the pre-condition at that AP is re-checked before executing any contingent procedure for forward recovery. If the pre-condition is violated, APRollback is performed to compensate the entire process.

CREATE RULE	ruleName::{pre | post | cond}
EVENT	apID(apParameters)
CONDITION	rule condition specification
ACTION	action 1
[ON RETRY	action 2]

Figure 1. Structure of an Integration Rule in an AP (Shrestha 2010)

Although the AP model provides flexible combinations of forward and backward recovery actions to handle internal errors and failure, the model does not provide approaches to respond to external events. The concept in (Shrestha 2010) also does not consider the potential inconsistencies that may arise in concurrent process execution due to data dependencies on the recovering process. The proposed research will extend the AP model to handle external events through AERs, and integrate the data dependency analysis capability into the extended model such that the recovery of a failed process includes the identification and interruption of the dependent processes.

4. Statement of Objectives

The main objective of this research is to develop the concept of Application Exception Rules to support flexible exception handling and response to external events, and integrate their use with support for analysis of data dependencies in concurrent process execution. The use of AERs will enable processes to handle exceptions and respond to external events based on their execution status. Such responses to failures and exceptions will involve the recovery of the process to a consistent state based on the techniques proposed in (Shrestha 2010). In the proposed research, the recovery of a process will be integrated with the additional ability to perform data dependency analysis. This will enable the identification of, and the communication of recovery to concurrently running processes that may be affected by the recovery of a process. The proposed work therefore involves developing the AER concept, along with the capability to perform data dependency analysis.

This research will demonstrate the use of Application Exception Rules through the integration and extension of the concepts proposed in (Shrestha 2010) and (Liu 2009). In the proposed environment, when a process receives an external event,

- the corresponding AER will be triggered.
- based on the most recently crossed AP, a recovery action will be fired. This will enable the process to recover itself to a consistent state depending on its current execution status.
- data dependency analysis will be performed to determine which processes are dependent only on the "recovered portion" of the current process.
- the process execution interruption and recovery will be communicated to the identified dependent processes in the form of events.

In the proposed concept, Application Exception Rules will use a case based rule structure to provide variability of response to exceptions. These rules, in association with assurance points, define the responses to the exceptions depending on the process execution status. Using this concept, different instances of a process may respond to events differently depending on the APs they have passed during execution. Each AER defines recovery actions based on APs. Therefore, when an exception triggers an Application Exception Rule, it fires a recovery action depending on the most recently passed assurance point. For example, if a process receives an event about the failure and recovery of another process that it is dependent on, the current process, using AERs, may avoid inconsistency by recovering itself based on the APs that it has passed. Thus, the proposed concept will enhance the exception handling mechanisms to also consider critical points in process execution that may affect recovery actions, and support variable response to exceptions.

To achieve the main objective, this research will involve the following sub-objectives.

a) Extend the AP model to support the specification of Application Exception Rules.
b) Integrate the use of data dependency analysis with the extended AP model.
c) Design and prototype an execution environment with data dependency analysis capability to demonstrate the AER concept.
d) Demonstrate and evaluate the Application Exception Rules model for concurrent process execution with a sample processing scenario.

5. Discussion of Objectives

This section provides a more detailed discussion of the objectives of this research outlined in the previous section.

a) Extend the AP model to support the specification of Application Exception Rules.

AERs respond to exceptional conditions based on the execution status of the corresponding instance of the process. To determine the execution status of a process instance, it will be necessary to store information about critical points of execution that it has passed. The concept described in (Shrestha 2010) uses APs as logical checkpoints where critical data related to the current execution status of a process is stored. Therefore, developing the AER concept will involve the checkpointing functionality of the APs. This research will extend the constraint checking and failure recovery functionality of the AP model to include exception handling by the specification of AERs.

Since, the AER concept needs to enable responding to exceptions variably, according to the execution status of the process, each AER will have a case structure and define recovery actions based on APs, as shown in the middle column of Figure 2. In the structure of an AER, each AP represents the fact that the process has crossed a certain critical point of execution. Responding to an exception depends on APs that have been passed by different instances of the process. When an exception occurs, it will get communicated to the process as an event. This event will trigger an AER. The triggered rule, based on the AP that was most recently passed by the corresponding process instance, will carry out the recovery actions according to the case structure.

As part of developing a prototype system, this research will use XML to specify Application Exception Rules for concurrently running processes in a sample scenario. The *db4o* (database for objects) object-oriented database will be used to store information about process execution and AP information.

b) Integrate the use of data dependency analysis with the extended AP model.

The proposed research involves the use of AERs of a process to handle exceptions and at the same time maintain data consistency in the concurrent process execution environment by informing dependent processes of data modifications that may affect them. Therefore, the research work will involve the provision of the data dependency analysis capability. This functionality will be built primarily from the concept proposed in (Liu 2009) to determine data dependencies. As described in Section 3.1, each site of execution will have a PEXA agent that maintains a local delta object schedule. A similar approach will be followed in the AER system.

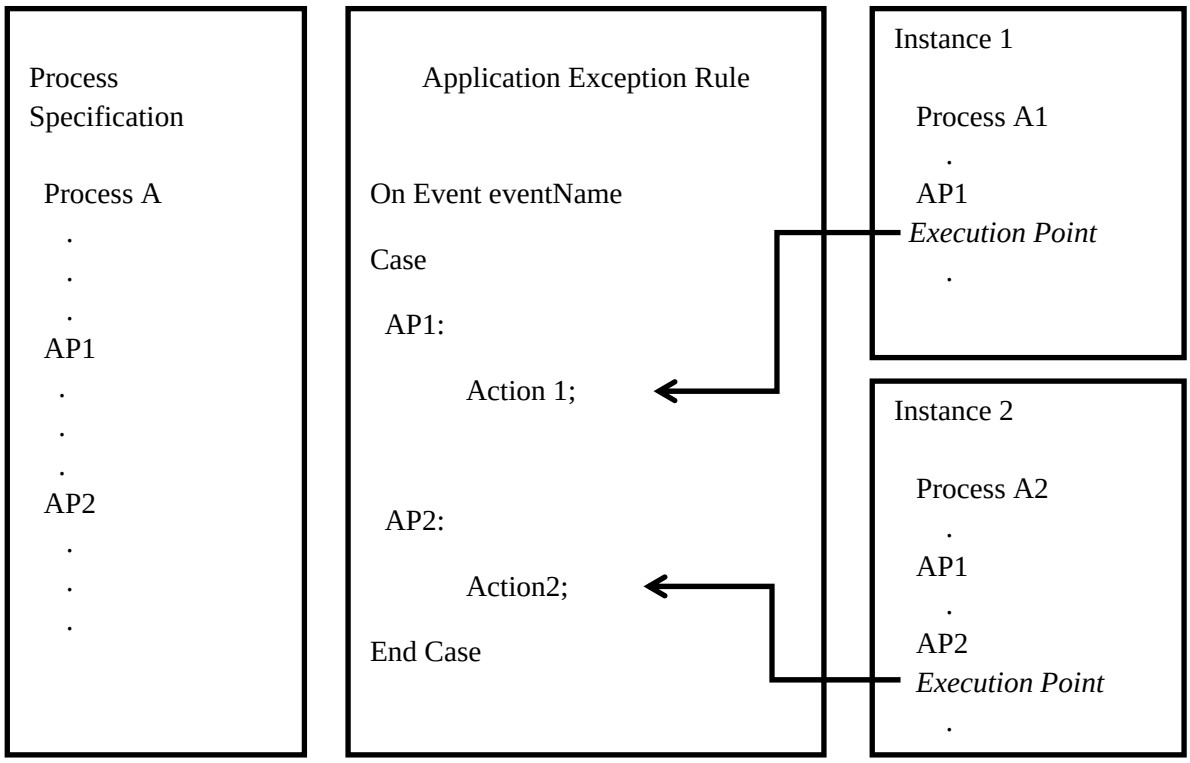

Figure 2: The Use of Application Exception Rules

The work in (Liu 2009) provides the Lazy and Eager algorithms to determine data dependencies. It also assumes that whenever a process fails, it recovers itself by rolling back completely. In such cases, any process that is read/write dependent on the main process is included in the list of dependent processes. That is, the dependencies are determined on a *process level*. But in the proposed research, a failed process will be recovering itself only to a consistent state of execution. Therefore, only processes that are dependent on the *recovered portion* of the main process need to be identified and informed.

To address this issue, the following steps will be taken.

- Since data dependency analysis needs to be handled only on the recovered portion of the process, the Lazy approach will be used as a preferred choice. The Eager approach assumes that all processes will potentially fail and determines dependent processes dynamically to support fast recovery. But since recovery in that context means rollback, it will not be suitable for the AER system which will support flexible process recovery to critical and consistent execution states. On the other hand, the Lazy approach determines the dependencies only when a process fails. This will be more suitable to the proposed system. A critical difference here would be that the list of dependent processes will be built only after the process is recovered to a consistent state. This will enable the availability of information about the "*start and end points of recovery*". Based on these points, the lazy approach will be employed to identify dependent processes.

- The data dependency analysis capability of the AER system will also require modifications to the types of information stored in the delta object schedule and the way in which the delta object schedules are queried to retrieve information about write dependent and potentially read dependent processes. In the general approach (Xiao, and Urban 2008), when a process fails, dependent processes are determined by querying the delta object schedule to identify processes that have a write/read operation after a write operation by that process. But in the AER system, since only the recovered portion of the process needs to be considered, it will be necessary to modify the delta object schedule to also include information about APs crossed in execution. This way, the querying of the schedule can be modified such that only the dependencies of a process between two particular APs (start and end points of recovery) are identified.

c) *Design and prototype an execution environment with data dependency analysis capability to demonstrate the AER concept.*

As part of the research, an execution environment in JAVA with support for concurrent process execution will be designed and implemented. The prototype will be developed as an extension to the AP model with support for the specification and functionality of AERs. In addition, the execution environment will be designed to support the use of Delta Enabled Grid Services (DEGS) (Urban et al., 2009) to facilitate the analysis of data dependencies between the concurrently running processes. The proposed system will be designed such that there is an associated PEXA with the execution environment which maintains the local delta object schedule in a time-sequenced manner. Each process in the prototype will be designed to handle exceptions. When the occurrence of an exception triggers an AER, leading to the execution of a recovery action for the process, construction of the process dependency graph to determine dependent process will also be done immediately. After the dependent processes identification is handled, steps to generate events to interrupt their execution will be followed.

d) *Demonstrate and evaluate the Application Exception Rules model for concurrent process execution with a sample processing scenario.*

The proposed research will develop the AER concept, integrate it with the AP model, and extend it to have data dependency analysis capability. The research will also involve testing and demonstrating the AER system in a concurrent process execution scenario. After the execution environment is prototyped, the demonstration of the sample application will be used to test the functionality of the system.

6. Timeline for the Research

A tentative schedule for the research is given below.

Milestone	Start Date	End Date
Review of past and current related work	Jun 2010	Aug 2010
Definition of research direction	Aug 2010	Sep 2010
Thesis proposal	Sep 2010	Nov 2010

Extension of AP model for AER specification	Oct 2010	Nov 2010
Integration of operation-level data dependency analysis	Nov 2010	Dec 2010
Development of execution environment for the AER system	Dec 2010	Jan 2011
Demonstration and evaluation of the AER system	Jan 2011	Mar 2011
Initial writing of thesis	Jan 2011	Feb 2011
Submission of thesis outline for first review	-	Feb 2011
Submission of thesis to committee for final review	-	Mar 2011
Thesis Defense	-	Mar 2011
Graduation	-	May 2011

7. Summary

This proposal has defined the concept of application exception rules together with the use of data dependency analysis. The proposed research will use AERs to provide an efficient and flexible way of responding to external events. The use of AERs will enable processes to give variable response to exceptions, depending on their execution status. This provides an approach to dynamically handle exceptions without giving a fixed response to each exception.

The proposed work will also employ the use of AERs in a concurrent process execution scenario with support for data dependency analysis. Therefore, the flexible exception handling and failure recovery capabilities of the AER model will include the identification of processes dependent on the recovered portion of a failed or interrupted process. These dependent processes will be informed of the recovery by interruption through events. Thus, this research will provide an approach for exception handling with strong support for data consistency for concurrent process execution through AERs and data dependency analysis.

References

Baresi, L. and Guinea, S. (2005). Towards Dynamic Monitoring of WS-BPEL processes. *Proceedings of the 3rd International Conference on Service Oriented Computing (ICSOC'05)*, Amsterdam, Netherlands.

Brambilla, M., Ceri, S., Comai, S., and Tziviskou, C. (2005). Exception handling in Workflow-Driven Web Applications. *Proceedings of the 14th international conference on World Wide Web*, 170-179.

Breugel, F. V., and Koshkina, M. (2006). *Models and Verification of BPEL*. Technical Report, York University, Toronto, Canada.
http://www.cse.yorku.ca/~franck/research/drafts/tutorial.pdf

Chakravarty, P., and Singh, M. P. (2007). An Event-Driven Approach for Agent-Based business Process Enactment. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '07).*

Chan, K. S. M., Bishop, J., Steyn, J., Baresi, L., and Guinea, S. (2007). A Fault Taxononmy for Web Service Composition. *Lecture Notes in Computer Science* In *Proceedings of the 3rd International Workshop on Engineering Service Oriented Applications (WESOA '07)*, Springer, 4907: 363-375.

Charfi, A. and Mezini, M. (2006). Aspect-Oriented Workflow Languages. *Proceedings of Conference on Cooperative Information Systems (CoopIS 2006).*

Charfi, A. and Mezini, M. (2007). AO4BPEL: An Aspect-Oriented Extension to BPEL. *World Wide Web Journal*, 10, 3: 309-344.

Christos, K., Costas, V., and Panayiotis, G. (2007). Enhancing BPEL scenarios with Dynamic Relevance-Based Exception Handling. *IEEE International Conference on Web Services (ICWS 2007)*, 751-758.

Eder, J. and Liebhart, W. (1996). Workflow recovery. *Proceedings of the First IFCIS International Conference on Cooperative Information Systems*. 124-134.

Elmasri, and R. Y., Navathe, S. B. (2010). *Fundamentals of Database Systems* (6th ed.): Addison-Wesley Longman Publishing Co., Inc.

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

Foster, H., Uchitel, S., Magee, J., 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 (ICSE 2006)*.

Fu, X., Bultan, T., and Su, J. (2004). WSAT: A Tool for Formal Analysis of Web Services. In *Proc. 16th Int. Conf. on Computer Aided Verification*, 3114: 510-514.

Greenfield, P., Fekete, A., Jang, J., and Kuo, D. (2003). Compensation is not enough. In *Proceedings of the 7th International Conference on Enterprise Distributed Object Computing (EDOC)*, 232.

Greenfield, P., Fekete, A., Jang, J., Kuo, D, and Nepal, S. (2007). Isolation Support for Service-based Applications: A Position Paper. *Proceedings of the 3rd Biannual Conference on Innovative Database Systems Research (CIDR '07)*, 314-323.

Huhns, M. N. (2002). Agents as Web Services. *IEEE Internet Computing*, 6, 4: 93-95.

Jang, J., Fekete, A., and Greenfield, P. (2007). Delivering Promises for Web Service Applications. *IEEE International Conference on Web Services (ICWS '07)*, 599-606.

Jennings, N. R., Faratin, P., Norman, T. J., O'Brien, P., Odgers, B., and Alty, J. L. (2000). Implementing a Business Process Management System using ADEPT: A Real-World Case Study. *Applied Artificial Intelligence*, 14, 5: 421-463

Jin, Y. (2004). *An architecture and Execution Environment for Component Integration Rules:* Ph.D. Dissertation, Arizona State University.

Kiczales, G., Lamping, J., Mendhekar, A, et al. (1997). Aspect-Oriented Programming. *Lecture Notes in Computer Science* In *Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP)*, Springer, 1242: 220-242.

Limthanmaphon, B., and Zhang, Y. (2004). Web service composition transaction management. In *Proceedings of the 15th Australasian Database Conference (ADC 2004), Conference in Research and Practice in Information Technology*, 27: 171-179.

Liu, Z. (2009). *Decentralized Data Dependency Analysis for Concurrent Process Execution*. M.S. Thesis, Texas Tech University.

Liu, A., Li, Q., Huang, L., and Xiao, M. (2007). A Declarative Approach to Enhancing the Reliability of BPEL Processes. *Proceedings of the International Conference on Web Services*, 272-279.

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

Luo, Z., Sheth, A., Kochut, K., and Miller, J. (2000). Exception handling in workflow systems. *Applied Intelligence*, 13, 2: 125-147.

Michelson, B. M. (2006). Event-Driven Architecture Overview. *Patricia Seybold Group*.

Mikalsen, T., Tai, S., and Rouvellou, I. (2002). Transactional Attitudes: Reliable Composition of Autonomous Web Services. In *Proceedings of the Workshop on Dependable Middleware-Based Systems (WDMS 2002)*, 44-53.

Modafferi, S. and Conforti, E. (2006). Methods for Enabling Recovery Actions in Ws-BPEL. *Proceedings of the International Conference on Cooperative Information Systems (CoopIS)*, 4275: 219.

Modafferi, S., Mussi, E., and Pernici, B. (2006). SH-BPEL: A Self-Healing plug-in for Ws-BPEL Engine. In *Proceedings of the 1st Workshop on Middleware for Service Oriented Computing (MW4SOC)*, 48-53.

Müller, R., Grener, U., and Rahm, E. (2004). AgentWork: A Workflow System Supporting Rule-Based Workflow Adaptation. *Database and Knowledge Engineering*, 51, 2: 223-256.

Papazoglou, M. P., and Georgakopoulos, D. (2003). Service-Oriented Computing. *Communications of the ACM*, 46, 10: 25-28.

Shen, W., Ghenniwa, H., and Li, Y. (2006). Agent-Based Service-Oriented Computing and Applications. *Proceedings of the 1st International Symposium on Pervasive Computing and Applications*, 8-9.

Shrestha, R. (2010). *Using Assurance Points and Integration Rules for Recovery in Service Composition:* M.S. Thesis, Texas Tech University.

Tan, W., Fong, L., and Bobroff, N. (2007). BPEL4JOB: A fault-handling design for job flow management. *Proceedings of the 5th international conference on Service-Oriented Computing*, 4749: 27-42.

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. *Lecture notes in Computer Science, In Service Oriented Computing -- ICSOC 2005, Third International Conference*, Springer, 3826: 495.

Urban, S. D., Dietrich, S. W., Na, Y., Jin, Y., Sundermier, A., and Saxena, A. (2001). The IRules Project: Using Active Rules for the Integration of Distributed Software Components. *Proceedings of the 9th IFIP Working Conference on Database Semantics: Semantic Issues in E-Commerce Systems*, 265-286.

Urban, S. D., Xiao, Y., Blake, L., and Dietrich, S. W. (2009). Monitoring Data Dependencies in Concurrent Process Execution through Delta-Enabled Grid Services. *International Journal of Web and Grid Services, 5*, 1: 85-106.

Xiao, Y., and Urban, S. D. (2008). 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. D. (2009). The DeltaGrid Service Composition and Recovery Model. *International Journal of Services Research (IJWSR)*, 6, 3: 35-66.

Zhao, W., Moser, L. E., and Melliar-Smith, P. M. (2005). A Reservation-Based Coordination Protocol for Web Services. *In Proceedings of the IEEE International Conference on Web Services (ICWS'05)*, 49-56.