The Dynamics of Process Modeling: New Directions for the Use of Events and Rules in Service-Oriented Computing

Susan D. Urban, Le Gao, Rajiv Shrestha, and Andrew Courter

Texas Tech University
Edward E. Whitaker Jr. College of Engineering
Department of Computer Science
Lubbock, TX 79409
{susan.urban,le.gao}@ttu.edu

Abstract. The introduction of service-oriented computing has created a more dynamic environment for the composition of software applications, where processes are affected by events and data changes and also pose data consistency issues that must be considered in application design and development. This chapter addresses the need to develop a more effective means to model the dynamic aspects of processes in contemporary, distributed applications, especially in the context of concurrently executing processes that access shared data and cannot enforce traditional transaction properties. After an assessment of current tools for process modeling, we outline four approaches for the use of events and rules to support dynamic behavior associated with constraint checking, exception handling, and recovery. The specific techniques include the use of *integration rules*, *assurance points*, *application exception rules*, and *invariants*. The chapter concludes with a discussion of future research directions for the integrated modeling of events, rules, and processes.

Keywords: service composition, event and rule processing, integration rules, application exception rules, assurance points, invariant conditions, dynamic process modeling

1 Introduction

The advent of Web Services and service-oriented computing has significantly changed software development practices and data access patterns for distributed computing environments, creating the ability to develop processes that are composed of service executions. These processes are often collaborative in nature, involving long-running activities based on loosely-coupled, multi-platform, service-based architectures. This new software development paradigm makes the concept of virtual organizations a reality, better supporting enterprise-to-enterprise business processes and data exchange. Service-oriented computing, however, also poses new challenges for software process modeling. In particular, processes must

be flexible enough to respond to the different types of change that can occur during execution. Change occurs, for example, when exceptional conditions arise in an application, such as a customer canceling or changing an order, or a warehouse discovering damaged shipments. Change also occurs when a process fails and needs to be recovered in a manner that maintains consistency for the failed process as well as for other processes that access shared data with the failed process. Processes must also be capable of executing in environments that no longer support traditional transactional properties but also guarantee correctness and consistency of execution. The ability to respond to change and guarantee consistency requires not only a flexible execution environment, but also techniques that support the modeling of a process's ability to correctly respond to events and failures that affect the normal flow of execution.

In this chapter, we first summarize existing techniques for modeling the dynamics of processes. We then introduce additional considerations for the use of events and rules in process modeling, especially in the context of service-oriented computing. In particular, this chapter illustrates the use of *integration rules*, *invariant rules*, and *application exception rules* together with the concept of *assurance points* to model the more dynamic nature of service-oriented computing. Integration rules are similar to the use of events and rules to control process flow [11, 12, 20, 38]. They are different, however, in that events are raised before and after the execution of services to trigger integration rules that test control logic that is orthogonal to the main procedural specification of process flow. Assurance points (APs) enhance the use of integration rules, providing checkpoints that are placed at critical locations in the flow of a process. An AP is used to store execution data that is passed as parameters to integration rules that check pre and post conditions and invoke additional execution logic. APs are also used as intermediate rollback points to support compensation, retry, and contingent procedures in an attempt to maximize forward recovery.

Whereas integration rules can be used to check data conditions at certain points in process execution, invariants provide a stronger way to monitor data conditions that must hold over a certain period of time during the execution of a process, especially when data items cannot be locked over the span of multiple service executions. Invariants are activated with a starting AP, deactivated with an ending AP, and monitor the data of the invariant condition in between APs using a concept known as Delta-Enabled Grid Services (DEGS) [2]. 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.

Finally, application exception rules provide a way 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 as determined by assurance points. Application exception rules can also be combined with a data dependency analysis procedure associated with the use of DEGS to provide a way to help a process determine how its own recovery or forward execution

can be affected by the failure and recovery of other processes that are accessing shared data [46]. This is especially important for maintaining data consistency in environments that cannot provide traditional transaction processing guarantees.

In the sections that follow, we first outline past work in the area of process modeling with a specific focus on the use of events, rules, and exception handling to provide dynamic behavior. We then provide motivation for integration rules, assurance points, invariants, and application exception rules in the context of decentralized process execution agents (PEXAs) that we have developed as part of our research on service-oriented computing. We then elaborate on assurance points and the rule functionality of our research. The chapter concludes with a summary and discussion of future research for modeling methodologies, modeling tools, and execution environments that support events and rules.

2 Conceptual Modeling of Business Processes

As described by Lu and Sadiq [26], most modeling techniques can be categorized as either graph-based techniques or rule-based techniques. The following subsections summarize graph and rule-based techniques, with a focus on support for dynamic capabilities.

2.1 Graph-Based Modeling Techniques

In graph-based modeling techniques such as BPMN [43], UML [14], and EPC [36], a business process is described by a graph notation in which activities are represented as nodes, and control flow and data dependencies between activities as arcs or arrows

BPMN. The Business Process Modeling Notation (BPMN V1.0) was introduced by the Business Process Management Initiative in 2004 [43]. The objective of BPMN is to provide a graphical model that can depict business processes and can be understood by both users and developers. Flow objects include symbols to represent events, activities, and gateways (i.e., decision points). Flow objects are connected to each other via connecting objects that represent sequence flow, message flow, and association. A process always starts from an event and ends in an event. All other events inside the process are called intermediate events and can be part of the normal flow or attached to the boundary of an activity. An attached event indicates that the activity to which the event is attached should be interrupted when the event is triggered. The attached event can trigger either another activity or sub-process. Typically, error handling, exception handling, and compensation are triggered by the attached event.

To detail a business process, swim lanes and artifacts can be used. Swim lanes are used to either horizontally or vertically group a process into subgroups by rules, such as grouping processes by departments in a company business process. Artifacts provide additional information in a business process to make a model more readable, such as text descriptions attached to an activity.

BPMN (V2.0 beta 1) [3] was released in 2009. In BPMN 2.0, the most important update is standardized execution semantics which provide execution semantics for all BPMN elements based on token flows. A choreography model is also supported in BPMN 2.0. Other significant changes include 1) a data object supporting assignments for activity; 2) updated gateways supporting exclusive/parallel event-based flow; 3) event-subprocesses used to handle event ocurrences in the bounding subprocess; 4) a call activity type that can call another process or a global task; and 5) escalation events for transferring control to the next higher level of responsibility.

Mapping tools that can convert BPMN to executable languages, where the translation is enhanced with the execution semantics of BPMN 2.0. For example, the Business Process Execution Language (BPEL) [1] is used for executing business processes that are composed of Web Services. A well-known, open-source mapping tool is BPMN2BPEL [31].

UML. The Unified Modeling Language (UML) is a general-purpose modeling language with widespread use in software engineering. UML provides a set of graphical modeling notations to model a system. An activity diagram describes a business process in terms of control flow. A state diagram represents a business process using a finite number of states. UML also provides sequence diagrams that emphasize interactions between objects.

In UML 2.0, new notations have been added to activity diagrams to provide support for the specification of pre and post conditions, events and actions, time triggers, time events, and exceptions. These notations provide more dynamic support to process modeling in UML. Researchers have also proposed process modeling enhancements to UML. For example, the work in [32] proposes a framework that supports exception handling using UML state charts. In [15], the authors present a method that can handle exceptions in sequence diagrams.

EPC. The Event-driven Process Chain (EPC) method was developed within the framework of the Architecture of Integrated Information Systems (ARIS) in the early 1990s. The merit of EPC is that it provides an easy-to-understand notation. OR, AND, and XOR nodes are used to depict logical operations in the process flow. The main elements of a process description include events, functions, organization, and material (or resource) objects. The EPC does not have specific notation support for exception processing. Instead, EPC uses the logical operations to specify the handling of events and exceptional conditions. Recent work has modified the EPC notation to provide better support for process modeling. For example, in [29], yEPC provides a cancellation notation to model either an activity or a scope cancellation process.

Other Related Methods. FlowMake is presented by Sadiq and Orlowska in [35]. FlowMake models a workflow using a graphical language, including workflow constraints that can be used to verify the syntactic correctness of a graphical workflow model. Reichert and Dadam [34] present a formal foundation for the support of dynamic structural changes of running workflow instances. ADEPTflex is a graph-based modeling methodology that supports users in modifying the structure of a running workflow, while maintaining its correctness and con-

sistency [34]. YAWL [42] is designed based on Petri nets for the specification of control flow. ActivityFlow [24] provides a uniform workflow specification interface to describe different types of workflows and helps to increase the flexibility of workflow processes in accommodating changes. ActivityFlow also allows reasoning about correctness and security of complex workflow activities independently from their underlying implementation mechanisms.

An advantage of graph-based languages is that they are based on formal graph foundations that have rich mathematical properties. The visual capabilities also enhance process design for users and designers. The disadvantage is that graph-based modeling methods are not agile for dynamic runtime issues, requiring the use of specialized notations that can cause the model to become more complex.

2.2 Rule-Based Modeling Techniques

In a rule-based modeling approach, business rules are defined as statements about guidelines and restrictions that are used to model and control the flow of a process [16]. More recently, rules are used together with agent technology to provide more dynamic ways of handling processes.

Use of Rules in Workflow and Service Composition: Active databases extend traditional database technology with the ability to monitor and react to circumstances that are of interest to an application through the use of Event-Condition-Action (ECA) rules [44].

The work of Dayal et al. [9] was one of the first projects to use ECA rules to dynamically specify control flow and data flow in a workflow. In the CREW project [21], ECA rules are used to implement control flow. The TrigSFlow [23] model uses active rules to decide activity ordering, agent selection, and worklist management. Database representation of workflows [17] uses Event-Condition-Message rules to specify workflows and utilize database logging and recovery facilities to enhance the fault-tolerance of the workflow application. Migrating workflows[7] provide dynamics in workflow instances. A migrating workflow transfers its code (specification) and its execution state to a site, negotiates a service to be executed, receives the results, and moves on to the next site [7]. ECA rules are used to specify the workflow control.

Active rules also provide a solution for exception handling in workflow systems. ADOME [5] and WIDE [4] are commercial workflow systems that use active rules in exception handling. Rules are also used in workflow systems to respond to ad-hoc events that have predefined actions. Other workflow projects that use active rules are described in [6, 13]. Active rules have been used to generate data exchange policies at acquaintance time among peer databases [22].

Agent-Based Techniques. Agent technology has been introduced to model business processes. Agents are autonomous, self-contained and capable of making independent decisions, taking actions to fulfill design goals and to model elements in a business process. Agents also support dynamic and automatic workflow adaptations, thus providing flexibility for unexpected failures. ADEPT [18] is an agent-based system for designing and implementing processes. The

process logic is defined by a service definition language, where agents have sufficient freedom to determine which alternative path should be executed at runtime. AgentWork [30] is a flexible workflow support system that provides better support for exception handling using an event monitoring agent, an adaptation agent, and a workflow monitoring agent. Events represent exceptional conditions, with rule conditions and actions used to correct the workflow. The adaptation agent performs adjustments to the implementation. The workflow monitoring agent checks the consistency of the workflow after adaptation implementation. If the workflow is inadequate, the workflow monitoring agent will re-estimate the error and invoke a re-adaptation of the workflow.

Rule and agent-based modeling methods provide better support for flexibility and adaptability in process modeling. Rule-based methods support modifications at runtime much easier than graph-based methods. It is easy to modify a process model by rule-based methods, and, unlike graph-based methods, rule-based methods do not need new notations to express exception handling processes. Rule-based methods, however, can be difficult to use and understand.

3 Motivation for New Rule Functionality

As illustrated in the previous section, most process modeling techniques are aligned with either a procedural approach, specified as a flow graph, or a rule-driven approach, where events and rules are used to control the flow of execution. Rules provide a more dynamic way to respond to events that represent a need to change the normal flow of execution. The use of rules in process modeling is especially important considering the growing prevalence of complex events, event-driven applications, and business activity monitoring.

In our view, a dynamic approach to process modeling for service-oriented environments requires a combination of graph and rule-based techniques, where graph-based techniques provide a means for specifying the main application logic and events are used to interrupt or branch off of the main flow of execution, triggering rules that check constraints, respond to exceptions, and initiate parallel activity. Events and rules should also play an increased role in supporting failure and recovery activity. Planning for failure and recovery should be an integral component of process modeling for service-oriented architectures, especially in the context of concurrently executing processes that access shared data and cannot enforce traditional transactional properties.

Our research addresses consistency checking as well as failure and recovery issues for service-oriented environments through the use of integration rules, invariants, and application exception rules, used together with a checkpointing concept known as assurance points. To motivate the use of these concepts, consider a decentralized execution environment consisting of Process Execution Agents (PEXAs) as shown in Figure 1. In our research, a PEXA is responsible for monitoring the execution of different processes. As shown in Figure 1, PEXA 1 is responsible for the execution of P1 and P4, PEXA 2 is responsible for P2, and PEXA 3 is responsible for P3. Each process invokes services at various locations

within the network. As shown in Figure 1, P1 invokes operation_a at the site of PEXA 1, operation_b and operation_c at the site of PEXA 2, and operation_d at the site of PEXA 3.

Figure 1 also illustrates that PEXAs are co-located with Delta-Enabled Grid Services (DEGS). A DEGS is a Grid Service that has been enhanced with an interface that stores the incremental data changes, or *deltas*, that are associated with service execution in the context of globally executing processes [2, 41]. A DEGS uses an OGSA-DAI Grid Data Service for database interaction. The database captures deltas using capabilities provided by most commercial database systems. Our own implementation has experimented with the use of triggers as a delta capture mechanism, as well as the Oracle Streams capability [41]. Oracle Streams is a feature that monitors database redo logs for changes and publishes these changes to a queue to be used for replication or data sharing.

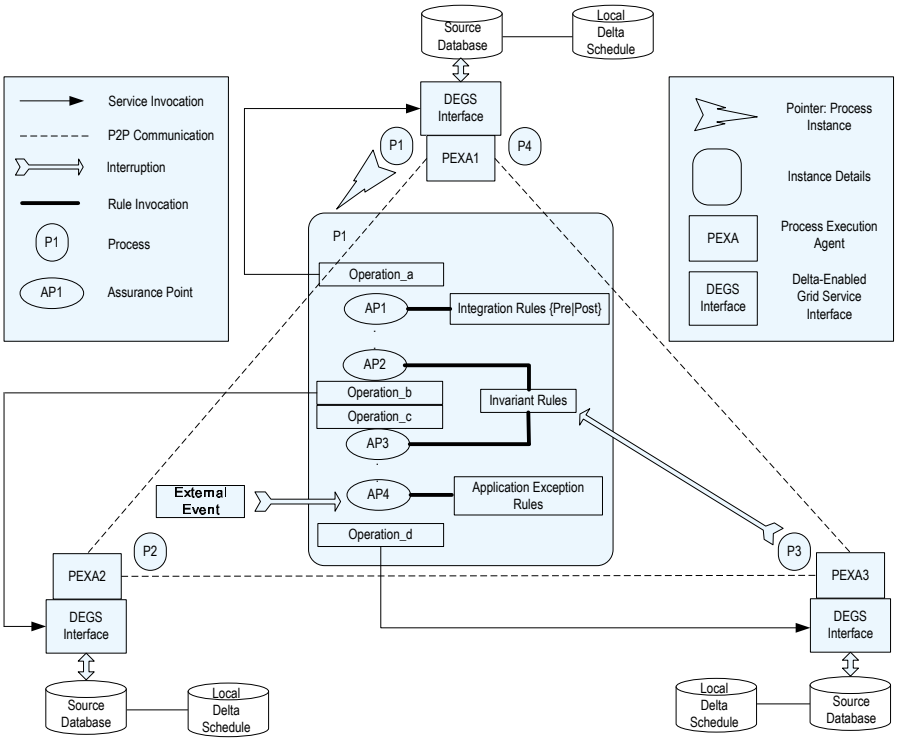

Fig. 1. Decentralized Process Execution Agents with Events and Rules

Deltas captured using DEGS are stored in a delta repository that is local to the service. Our past work has experimented with the creation of a centralized Process History Capture System (PHCS) that included deltas from all DEGSs in the environment and the process runtime context generated by the process

execution engine. Deltas are dynamically merged using timestamps as they arrive in the PHCS to create a time-ordered schedule of data changes from the DEGS. The global delta object schedule is used to support recovery activities when process execution fails [47, 46, 45], where the global delta object schedule provides the basis for discovering data dependencies among processes. Our most recent work has transformed the global delta object schedule into a distributed schedule with a decentralized algorithm for discovering data dependencies [40, 25]. As a result, each PEXA in Figure 1 has its own local delta object schedule.

Given that processes can execute in an environment where decentralized PEXAs can monitor data changes and communicate about data dependencies among concurrently executing processes, our work is focused on how to enhance the ability to monitor data consistency in the failure and recovery process. To illustrate our approach, the expanded view of P1 in Figure 1 shows the use of assurance points (APs) and the different types of rules. As shown for P1, APs can be placed at strategic locations in a process, where an AP is a combined logical and physical checkpoint that can be used to store execution data, alter program flow, and support recovery activity. One use of an AP is to trigger integration rules as shown for AP1, where integration rules check pre and post conditions for service execution. By checking pre/post conditions, user-defined consistency constraints can validated, which is important since most service-oriented environments cannot rely on traditional notions of serializability to ensure the correctness of concurrently executing processes.

Another use of an AP is to activate invariant rules. Invariants indicate conditions that must be true during process execution between two different APs. As shown for P1 in Figure 1, an invariant is monitored during the execution between AP2 and AP3, where the invariant represents a data condition that is critical to the correct execution of P1. P1, however, may not be able to lock the data associated with the invariant during the service executions between AP2 and AP3. Given that DEGS can be used to monitor data changes, P1 can activate the invariant condition, but still allow concurrent processes to access shared data. P1 can then be notified if data changes violate the invariant condition. For example, if P3 modifies data associated with the invariant of P1, P1 can re-evaluate the invariant condition and invoke recovery actions if needed.

P1 also illustrates the use of application exception rules at AP4. A process should be capable of responding to external events that may affect execution flow. The response to the event, however, may depend on the current status of the process. For example, P1 may respond one way if the process has passed AP4, but may respond differently if the process is only at AP1. Application exception rules therefore provide a case-based structure that allows a process to use information about assurance points to provide greater flexibility in response to events. Furthermore, since PEXAs can communicate about data dependencies among concurrently executing processes, when a process P_j invokes recovery procedures in response to integration rules, invariant conditions, or application exception rules, event notifications can be sent through P2P communication to dependent processes that are controlled by other PEXAs. Application exception

rules can be used by a process P_i to intercept such events, determine how the failure and recovery of P_j potentially affects the correctness conditions of P_i, and respond in different ways depending on the AP status of the process.

4 Assurance Point and Rule Functionality

This section provides a more detailed description of the capabilities outlined in Section 3. The first subsection elaborates on foundational work with integration rules. The following subsections then address assurance points, invariant rules, and application exception rules.

4.1 Dynamic Behavior with Integration Rules

Integration Rules (IRules) were originally defined in [38] to investigate the middle-tier, rule processing technology necessary for the use of declarative, active rules in the integration of Enterprise Java Beans (EJB) components. Several different subcomponents to the IRules language framework have been defined, including the Component Definition Language (CDL) for defining a global object model of components and their relationships [11], the IRules Scripting Language (ISL) for describing application transactions (a BPEL-like language), the Event Definition Language (EDL) for defining events, and the Integration Rule Language (IRL) for defining active rules [11, 12, 38]. In this section, we focus on IRL and the functionality that it provides for dynamically testing the correctness of process execution. The remaining subsections then show modifications to IRL for additional dynamic modeling capabilities that address exception handling and the consistency of concurrent processes.

IRules are different from past work with the use of rules to control workflow in that they are integrated with the use of procedural specifications. Using IRules, the main logic of a process can be expressed using a modeling tool such as BPMN. It is assumed, however, that the start and end of a process generates *application transaction events*. The execution of individual services within a process also generates *method events* both before and after the execution of a service. IRules are then used to respond to application transaction events and method events, controlling rule actions together with the normal process flow using rule coupling modes from active database technology [44]. Integration rules can therefore be used to check pre and post conditions, to change the flow of execution, to spawn a new flow of execution that eventually joins the main flow, to defer a new flow of execution upon successful completion of the main flow, or to invoke a new, independent, parallel flow of execution in addition to the main flow.

The structure of an integration rule is shown in Figure 2. Events are generated before and after the execution of individual services and their enclosing processes by wrappers that coordinate rule and component execution. Rule conditions and actions can be enhanced with ec (for event/condition) and ca (for condition/action) coupling modes [20]. For example, the immediate synchronous mode implies that the main flow of execution (i.e., the one that generated the

event that triggered the rule) is halted while rule execution occurs. The immediate synchronous mode is therefore useful for checking preconditions before the execution of a service. The immediate asynchronous mode allows the main flow to continue during rule execution, with the rule executing in the same transactional framework as the process that triggered the rule. The deferred mode provides a way of triggering a rule that schedules the execution of a procedure at the end of the main procedural flow. The deferred mode is useful for scheduling the execution of a post condition that must be used to ensure data consistency at the end of a service execution. Alternatively, a post condition can be tested by triggering an integration rule with an immediate synchronous mode after the execution of a service. The decoupled mode is used to trigger the execution of a rule condition or action that executes in parallel with the main flow of execution as a separate transactional entity. The decoupled mode therefore provides a way to use rules for invoking procedures that involve business activity monitoring.

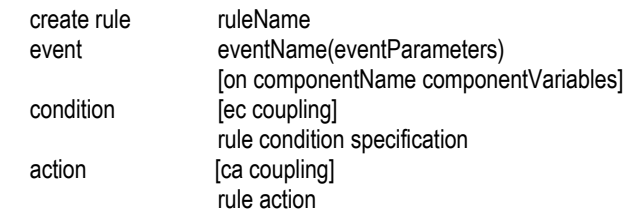

Fig. 2. Structure of an Integration Rule [20]

As an example, consider the integration rule for a stock application in Figure 3. The purpose of the stockSell rule is to initiate sellStock transactions when a price increase occurs. We only want to initiate such transactions, however, for pending orders where the NewPrice exceeds the desired selling price in the pending order. This situation implies that we need to compare the new price of the stock with the old price of the stock to determine if there was a price increase. This can only be done by examining the old and new values before the execution of the price change in the Stock component, thus illustrating the need for the beforeSetPrice event. The coupling mode on the rule condition is immediate, indicating that the check for a price increase should be performed as soon as the rule is triggered. The sellStock transactions on the appropriate pending orders, however, should only be executed after the completion of the setPrice method. As a result, the action part of the rule is deferred, meaning that the action will not be performed until the end of the outer-most transaction in which the rule was triggered.

The full details of the integration rule execution model as originally used with EJB components can be found in [19, 20, 38]. With respect to dynamic process modeling, integration rules illustrate the manner in which rules can be used for more than just the interconnection of steps in a workflow. Integration rules work

```
create rule     stockSell
event           beforeSetPrice(NewPrice)
                on stock S
condition       immediate
                when NewPrice>S.price
action          deferred
                from Pn in S.pendingTrades
                where S.price>=Pn.desiredPrice AND Pn.action="sell"
                do sellStock(S,Pn);
```

Fig. 3. Integration Rule Example for a Stock Application [38]

together with procedural, graph-based specifications and are particularly useful for 1) checking conditions that validate the correctness of service execution and 2) invoking business activity monitoring procedures that execute in parallel with business processes.

4.2 Dynamic Behavior with Assurance Points

In our current research, we have enhanced the use of integration rules using assurance points and recovery actions. An assurance point (AP) is defined as a process execution correctness guard as well as a potential rollback point during the recovery process [37, 39]. Given that concurrent processes do not execute as traditional transactions in a service-oriented environment, inserting APs at critical points in a process is important for checking consistency constraints and potentially reducing the risk of failure or inconsistent data. An AP also serves as a milestone for backward and forward recovery activities. When failures occur, APs can be used as rollback points for backward recovery, rechecking pre-conditions relevant to forward recovery. The work in [37, 39] has developed a prototype of APs using the Process Modeling Language (PML) described in [27, 28].

An AP is defined as: AP = <apId, apParameters*, IRpre?, IRpost?>, where:
- apID is the unique identifier of the AP
- apParameters is a list of critical data items to be stored as part of the AP,
- IRpre is an integration rule defining a pre-condition,
- IRpost is an integration rule defining a post-condition,
- IRcond is an integration rule defining additional application rules. In the above notation, * indicates 0 or more occurrences, while ? indicates zero or one optional occurrences.

IRpre, IRpost, and IRcond are expressed in the integration rule format introduced in Figure 2, where the eventName is the name of the assurance point that triggers the rule. For IRpre and IRpost, a constraint C is always expressed in a negative form (not(C)). The action of a rule is invoked if the pre or post condition is not true, invoking a recovery action or an alternative execution path. If the specified action is a retry activity, then there is a possibility for the process to execute through the same pre or post condition a second time. In such a case,

IRpre and IRpost rules support the specification of a second action to invoke a different recovery procedure the second time through.

In its most basic form, the recovery action of an integration rule simply invokes an alternative process. Recovery actions can also be one of the following actions:

- **APRollback**: APRollback is used when the entire process needs to compensate its way back to the start of the process.
- **APRetry**: APRetry is used when a process needs to be backward recovered using compensation to a specific AP. The backward recovery process will go to the first AP reached as part of the compensation process. The pre-condition defined in the AP is re-checked before resuming the execution.
- **APCascadedContingency (APCC)**: APCC is a backward recovery process that searches backwards through the hierarchical nesting of processes to find a contingent procedure for a failed sub-process. During the APCC backward recovery process, when an AP is reached, the pre-condition defined in the AP is re-checked before invoking a contingent procedure for forward recovery.

When the execution of a process reaches an AP, integration rules associated with the AP are invoked. The condition of an IRpost is evaluated first. If the post-condition is violated, the action invoked can be one of the pre-defined recovery actions as described above. If the post-condition is not violated, then an IRpre rule is evaluated before the next service execution. If the pre-condition is violated, one of the pre-defined recovery actions will be invoked. If the pre-condition is satisfied, the AP will check for any conditional rules (IRcond) that may exist. IRcond rules do not affect the normal flow of execution but provide a way to invoke parallel activity based on application requirements. Note that the expression of a pre-condition, post-condition or any additional condition is optional.

As an example, consider a subset of an online shopping process, as shown in Figure 4, where two APs are inserted. Both APs have integration rules that must be checked when the process execution reaches the APs. The cop and top in the process indicate the compensation and contingency of the attached activity, respectively. AP1 is orderPlaced, which reflects that the customer has finished placing the shopping order. Before executing the payment activity, the pre-condition at AP1 is checked to guarantee that the store has enough goods in stock. Otherwise, the process invokes the backOrderPurchase process instead. Similarly, the CreditCardCharged AP2 after payment activity has a post-condition that further guarantees that the in-stock quantity must be in a reasonable status (not less than zero) after the decInventory operation. Otherwise, a recovery action APRetry must be invoked to recover the process back to AP1 and re-execute the payment activity. If the post-condition fails after re-execution, then APRollback will be invoked to abort the overall process.

4.3 Dynamic Behavior with Invariants

APs together with integration rules allow data consistency conditions to be checked at specific points in the execution of a process [37], using rule actions

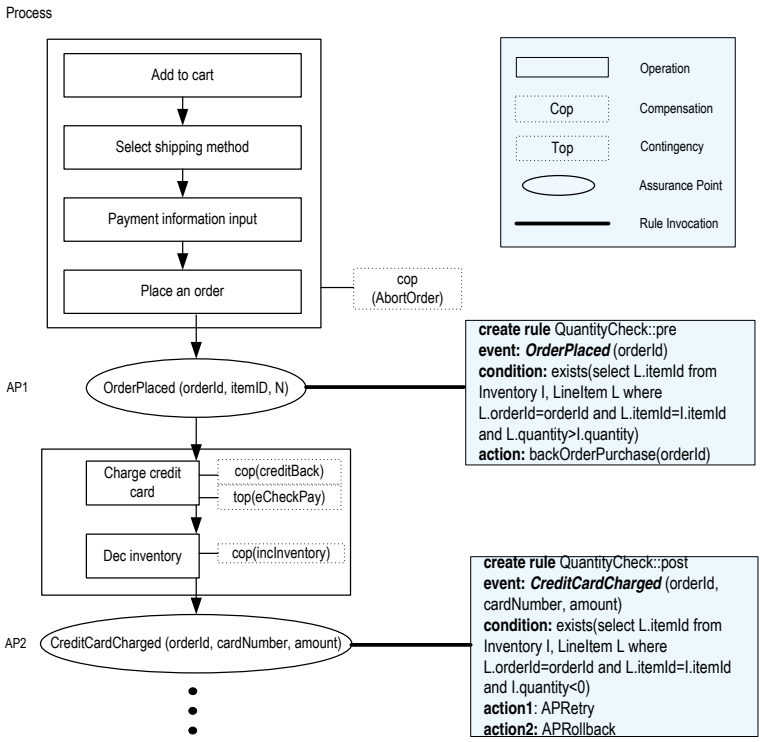

Fig. 4. Portion of a Process Illustrating Assurance Points and Integration Rules

to invoke recovery procedures. In some applications, however, stronger condition checking techniques may be needed to monitor data consistency. As a result, an additional way to use rules together with APs is through the use of invariants. An invariant is a condition that must be true during process execution between two different APs. An invariant is designed for use in processes where 1) isolation of data changes in between service executions cannot be guaranteed (i.e., critical data items cannot be locked across multiple service executions), and 2) it is critical to monitor constraints for the data items that cannot be locked. The data monitoring functionality provided by our previous work with DEGS makes it possible to monitor invariant conditions. Invariants provide a stronger way of monitoring constraints and guaranteeing that a condition holds for a specific duration of execution without the use of locking. A prototype of the invariant capability has been developed and demonstrated in [8].

Using the invariant technique, a process declares an invariant condition when it reaches a specific AP in the process execution, also declaring an ending AP for monitoring of the invariant condition. When a concurrent process modifies a data item of interest in an invariant condition, the process that activated the invariant is notified by a monitoring system built on top of Delta-Enabled Grid

Services. If the invariant condition is violated during the specified execution period, the process can invoke the recovery procedures defined in Section 4.2. The strength of the invariant technique is that it provides a way to monitor data consistency in an environment where the coordinated locking of data items across multiple service executions is not possible.

An invariant has an identifier, two AP specifications (AP_s as a starting AP and and AP_e as an ending AP), and optional parameters that are necessary in the condition specification. Once AP_s is reached, the invariant rule condition becomes active. The condition is specified as an SQL query. The condition is initially checked and the action is executed if the invariant condition is violated. If the invariant condition holds, the rule condition goes into monitoring mode using the DEGS capability. The condition monitoring continues until AP_e is reached or until the invariant condition is violated.

As shown in Figure 5, when an invariant condition goes into monitoring mode, the data items of interest in the invariant condition are registered with a monitoring service. The monitoring service subscribes to the DEGSs that contain the relevant data items referenced in an invariant. The DEGSs of the environment will notify the service of any changes to the relevant data items by concurrent processes. Any deltas that are forwarded to the monitoring service will cause the invariant condition to be rechecked. As long as the condition still holds, then there is no interference among the concurrent process executions. If the condition is violated, then the recovery action of the invariant rule will be executed.

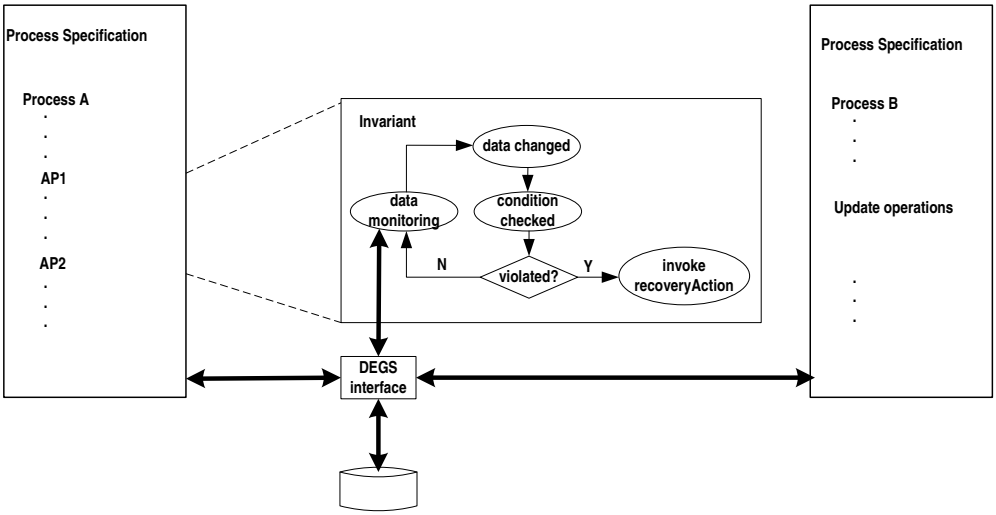

Fig. 5. Semantics of an Invariant under Monitoring Mode

As a specific example, consider the invariant in Figure 6, where the LoanAmount-Monitoring invariant is to be monitored between the LoanAppCreation AP (i.e., the starting AP for the monitoring process) and the LoanCompletion AP (i.e., the ending AP for the monitoring process). The process represents a loan approval process, where the process is creating a loan application for a customer at a bank that already has an account at that bank. Figure 6 shows an invariant that checks to make sure the loan applicant has a tenth of the requested loan amount in the account, where the amount and customerid are passed as parameters from the LoanAppCreation AP. The condition is expressed as an SQL query, preceded with the not exists clause. Therefore, according to the SQL condition defined, if the account balance does not meet the criteria, then the select condition will return no tuples, making the not exists clause true, which triggers recoveryAction1. If the query returns tuples that satisfy the SQL condition, then the process continues and the status of the SQL query is monitored using the DEGS capability and the invariant monitoring system. If the process reaches the LoanCompletion AP and the applicants account balance still meets the necessary criteria, then the process continues past the LoanCompletion AP, completing the loan application after deactivating the LoanAmountMonitoring invariant. If at anytime between the LoanAppCreation AP and the LoanCompletion AP, the applicant's account balance falls below the necessary criteria, the invariant monitoring system will notify the process, which will execute one of the recovery actions.

```
create rule          LoanAmountMonitoring::inv
event                LoanAppCreation(LoanCompletion, customId)
condition            (Not exists (select * from loan where loan.applicantID =
                       '"+customId+"' and loan.status='pre-qualified' and
                       loan.amount < (select 10*balance from account where
                       account.customId = '"+customId+"'))
recoveryAction1      APRetry
recoveryAction2      APRollback
```

Fig. 6. An Invariant Example for Monitoring a Bank Balance for a Loan Approval Process

4.4 Dynamic Behavior with Application Exception Rules

Dynamic behavior can also be achieved by using rules to respond to exceptional conditions, where exceptions are communicated as events that interrupt the normal flow of execution. Whereas past work generally provides a fixed response to exceptions, our work with application exception rules, provides a more flexible way of using rules to respond to exceptions. We are currently developing the application exception rule functionality in [33].

As with integration rules and invariants, application exception rules are also associated with assurance points. An outline of a process with APs is shown in the leftmost column of Figure 7, where two different APs are defined. Each AP represents the fact that a process has passed certain critical points in the execution and that responding to an exception depends on the APs that have been passed for individual instances of a process. Application exception rules are then written to respond to exceptions according to the AP status of a process.

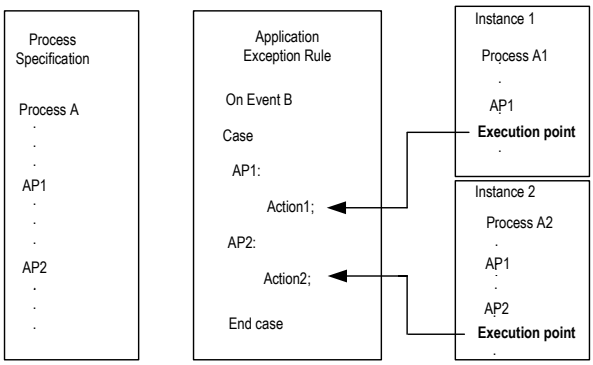

Fig. 7. The Use of Application Exception Rules

As shown in the middle column of Figure 7, application exception rules have a case structure, defining recovery actions based on APs. When an exception occurs, application exception rules are triggered. The exception handling procedure to execute varies according to the AP status of the process, where recovery actions can query the execution state associated with the most previous AP. As shown in Figure 7, one instance of process A executes recoveryAction1 since the process has passed AP1 but not AP2. The other instance of process A executes recoveryAction2 since the process has passed AP2. For example, in an order processing application, if an order is canceled before the packing and shipment of the order, then the order processing is cancelled. If the order is cancelled after the shipment has occurred, the order processing might be cancelled with an additional restocking fee charged to the customer.

APs and application exception rules represent the fact that a response to an exception is not always a fixed action. The manner in which a process responds to an exception depends on the state of the process. Identifying exceptions that alter the execution path should be a routine aspect of process modeling. Application exception rules advocate that the identification of exceptions should be extended to also consider the critical execution points that may affect recovery actions, and that rules, together with supportive execution environments, should be designed to provide variability of response.

A broader use of application exception rules is in the context of the decentralized execution environment with support for data dependency analysis as described in Section 3. Using the data monitoring capabilities of DEGS, we have developed a decentralized data dependency analysis algorithm [40, 25] to enhance recovery activities for concurrent processes that execute with relaxed isolation properties. In particular, when one process fails, recovery activities in the form of compensation can occur. Compensating procedures, however, may make changes to the data that has been read and used by concurrent processes. Using DEGS together with the decentralized data dependency analysis algorithm, the failure and recovery of one process can include the identification of other dependent processes that may be executing within the decentralized environment. Events can then be used to interrupt the execution of dependent processes, with application exception rules providing a way to respond to such events in a flexible manner.

We have experimented with this approach using process interference rules (PIRs) [46]. A PIR is written from the perspective of an executing process and is used to test user-defined correctness conditions to determine if a dependent process should continue running or invoke its own recovery procedures. We are currently integrating the PIR functionality into the concept of application exception rules to provide a more dynamic way to 1) recognize potential data inconsistency problems among concurrently executing processes and 2) use the event and rule functionality of application exception rules to interrupt dependent processes, test data consistency conditions, and invoke recovery procedures as needed.

5 Summary and Future Directions

This chapter has outlined several non-traditional uses of rules for supporting dynamic behavior in service-oriented environments. The advantage of the techniques presented is that they integrate the use of procedural and rule-based techniques for process modeling. As a proof of concept, prototypes have already been developed for integration rules, assurance points, the integration rule recovery actions, invariants, the process interference rule pre-cursor to application exceptions rules, and decentralized data dependency analysis [20, 37, 39, 40, 47, 25]. A prototype of the more general use of application exception rules is currently under development.

An interesting challenge lies in developing methodologies that are capable of supporting each rule paradigm in an integrated manner. Each rule type addresses a different dimension of dynamics for service-oriented environments. Methodologies are needed to define when and how the different rule forms are defined. Notational conventions are needed to enhance existing, graph-based approaches with notations that depict the way in which rules are used and integrated with procedural specifications. Guidelines are also needed to assist with the placement of assurance points, with the specification of the different types of rules, and with defining the conditions under which the rules are used. Execution

environments are also needed to support the dynamic capabilities supported by integration rules, application exception rules, and process interference rules. Our own research is focused on the development of decentralized Process Execution Agents that communicate in a peer-to-peer manner to dynamically determine data dependencies among concurrently executing processes and to coordinate the execution of processes with the different rule forms outlined in this chapter.

Acknowledgments. This research has been supported by NSF Grant No. CCF-0820152 and NSF Grant No. IIS-9978217.

References

1. Alves, A., Arkin, A., Askary, S., Barreto, C., Bloch, B., Curbera, F., Ford, M., Goland, Y., Guízar, A., Kartha, N., et al.: Web services business process execution language version 2.0. OASIS Standard http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html 11 (2007)
2. Blake, L.: Design and Implementation of Delta-Enabled Grid Services. M.S. Thesis, Deptment of Computer Science and Engineering, Arizona State Univ (2005)
3. BPMN, O.: BPMN 2.0 beta 1. http://www.omg.org/cgi-bin/doc?dtc/09-08-14.pdf (2009)
4. Ceri, S., Grefen, P., Sanchez, G.: WIDE-a distributed architecture for workflow management. In: proceedings of 7th Int. Workshop on Research Issues in Data Engineering. pp. 76–79 (1997)
5. Chiu, D., Li, Q., Karlapalem, K.: Exception handling with workflow evolution in ADOME-WFMS: a taxonomy and resolution techniques. ACM Siggroup Bulletin 20(3), 8 (1999)
6. Cichocki, A.: Workflow and process automation: concepts and technology. Kluwer Academic Pub (1998)
7. Cichocki, A., Rusinkiewicz, M.: Migrating workflows. NATO ASI series. Series F: computer and system sciences pp. 339–355 (1998)
8. Courter, A.: Supporting Data Consistency in Concurrent Process Execution with Assurance Points and Invariants. M.S. Thesis, Texas Tech University (2010)
9. Dayal, U., Hsu, M., Ladin, R.: A transactional model for long-running activities. In: Proceedings of the 17th International Conference on Very Large Data Bases. pp. 113–122. Citeseer (1991)
10. Desel, J.: Process modeling using petri nets. Process-Aware Information Systems: Bridging People and Software through Process Technology pp. 147–177 (2005)
11. Dietrich, S., Patil, R., Sundermier, A., Urban, S.: Component adaptation for event-based application integration using active rules. Journal of Systems and Software 79(12), 1725–1734 (2006)
12. Dietrich, S., Urban, S., Sundermier, A., Na, Y., Jin, Y., Kambhampati, S.: A language and framework for supporting an active approach to component-based software integration. Informatica-Ljubljana 25(4), 443–454 (2002)
13. Doğaç, A.: Workflow management systems and interoperability. Springer Verlag (1998)
14. Engels, G., Förster, A., Heckel, R., Thöne, S.: Process modeling using UML. Process Aware Information Systems: Bridging People and Software Through Process Technology pp. 85–118 (2005)

15. Halvorsen, O., Haugen, O.: Proposed notation for exception handling in UML 2 sequence diagrams. In: Software Engineering Conference, 2006. Australian. p. 10 (2006)
16. Herbst, H., Knolmayer, G., Myrach, T., Schlesinger, M.: The specification of business rules: A comparison of selected methodologies. In: Proceedings of the IFIP WG8. vol. 1, pp. 29–46. Citeseer (1994)
17. Jean, D., Cichock, A., Rusinkiewicz, M.: A database environment for workflow specification and execution. In: Proc. Intl Symposium on Cooperative Database Systems Kyoto (1996)
18. Jennings, N., Faratin, P., Norman, T., O?Brien, P., Odgers, B., Alty, J.: Implementing a business process management system using ADEPT: A real-world case study. Applied Artificial Intelligence 14(5), 421–463 (2000)
19. Jin, Y., Urban, S., Dietrich, S.: A concurrent rule scheduling algorithm for active rules. Data & Knowledge Engineering 60(3), 530–546 (2007)
20. Jin, Y., Urban, S., Dietrich, S., Sundermier, A.: An Integration Rule Processing Algorithm and Execution Environment for Distributed Component Integration. Informatica-Ljubljana 30(2), 193 (2006)
21. Kamath, M., Ramamritham, K.: Failure handling and coordinated execution of concurrent workflows. In: Proceedings of the International Conference on Data engineering. pp. 334–341. Citeseer (1998)
22. Kantere, V., Kiringa, I., Mylopoulos, J., Kementsietsidis, A., Arenas, M.: Coordinating peer databases using ECA rules. Databases, Information Systems, and Peer-to-Peer Computing pp. 108–122 (2004)
23. Kappel, G., Proll, B., Rausch-Schott, S., Retschitzegger, W.: TriGS/sub flow: Active object-oriented workflow management. In: Proc. of HICSS. p. 727. Published by the IEEE Computer Society (1995)
24. Liu, L., Pu, C.: Activity flow: Towards incremental specification and flexible coordination of workflow activities. Conceptual ModelingER'97 pp. 169–182 (1997)
25. Liu, Z.: Decentralized Data Dependency Analysis for Concurrent Process Execution. M.S. Thesis, Texas Tech University (2009)
26. Lu, R., Sadiq, S.: A survey of comparative business process modeling approaches. In: Business Information Systems. pp. 82–94. Springer (2007)
27. Ma, H.: The design and implementation of the GridPML: a process modeling language for the DeltaGrid. M.S. Thesis, Arizona State University (2005)
28. Ma, H., Urban, S., Xiao, Y., Dietrich, S.: GridPML: A Process Modeling Language and History Capture System for Grid Service Composition. Proceedings of the International Conference on e-Business Engineering (2005)
29. Mendling, J., Neumann, G., Nüttgens, M.: Yet another event-driven process chain. Business Process Management pp. 428–433 (2005)
30. Müller, R., Greiner, U., Rahm, E.: AW: a workflow system supporting rule-based workflow adaptation. Data & Knowledge Engineering 51(2), 223–256 (2004)
31. Ouyang, C., Dumas, M., Aalst, W., Hofstede, A., Mendling, J.: From business process models to process-oriented software systems. ACM transactions on software engineering and methodology (TOSEM) 19(1), 2 (2009)
32. Pintér, G., Majzik, I.: Modeling and analysis of exception handling by using UML statecharts. Scientific Engineering of Distributed Java Applications pp. 58–67 (2005)
33. Ramachandran, J.: Integrating Exception Handling and Data Dependency Analysis through Application Exception Rules. M.S. Thesis (in progress), Texas Tech University, to appear in (2011)

34. Reichert, M., Dadam, P.: ADEPT flexsupporting dynamic changes of workflows without losing control. Journal of Intelligent Information Systems 10(2), 93–129 (1998)
35. Sadiq, W., Orlowska, M.: On capturing process requirements of workflow based business information systems. In: Proceedings of the 3rd International Conference on Business Information Systems (BIS99) (1999)
36. Scheer, A., Thomas, O., Adam, O.: Process modeling using event-driven process chains. Process-aware information systems: bridging people and software through process technology pp. 119–145 (2005)
37. Shrestha, R.: Using Assurance Points and Integration Rules for Recovery in Service Composition. M.S. Thesis, Texas Tech University (2010)
38. Urban, S., Dietrich, S., Na, Y., Jin, Y., Saxena, S., Urban, S., Dietrich, S., Na, Y., Jin, Y.: The irules project: using active rules for the integration of distributed software components. In: Proceedings of the 9th IFIP 2.6 Working Conference on Database Semantics: Semantic Issues in E-Commerce Systems, Hong Kong. pp. 265–286. Citeseer (2001)
39. Urban, S., Gao, L.S., Courter, A.: Achieving Flexibility in Service Composition with Assurance Points and Integration Rules. To appear in Proc. of the Int. Conf. on Cooperative Information Systems, Crete, Greece, On The Move(OTM)Conferences, Part 1, Lecture Notes in Computer Science 6426, Springer, Heidelberg, pp. 28-437 (2010)
40. Urban, S., Liu, Z., Gao, L.: Decentralized data dependency analysis for concurrent process execution. In: Enterprise Distributed Object Computing Conference Workshops, 2009. EDOCW 2009. 13th. pp. 74–83 (2009)
41. Urban, S., Xiao, Y., Blake, L., Dietrich, S.: Monitoring data dependencies in concurrent process execution through delta-enabled grid services. International Journal of Web and Grid Services 5(1), 85–106 (2009)
42. Van Der Aalst, W., Ter Hofstede, A.: YAWL: yet another workflow language. Information Systems 30(4), 245–275 (2005)
43. White, S., et al.: Business Process Modeling Notation (BPMN) Version 1.0. Business Process Management Initiative, BPMI. org http://www. bpmi. org/bpmi-downloads/BPMN-V1. 0. pdf (2004)
44. Widom, J., Ceri, S.: Active database systems: Triggers and rules for advanced database processing. Morgan Kaufmann Pub (1996)
45. Xiao, Y., Urban, S., Liao, N.: The DeltaGrid abstract execution model: service composition and process interference handling. Conceptual Modeling-ER 2006 pp. 40–53 (2006)
46. Xiao, Y., Urban, S.: 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 (2008)
47. Xiao, Y., Urban, S.: The DeltaGrid Service Composition and Recovery Model. International Journal of Web Services Research 6(3), 35–66 (2009)