This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2824810, IEEE Transactions on Services Computing

Robust and Reliable Process-Aware Information Systems

André Luis Schwerz, Rafael Liberato, Calton Pu, *Fellow Member, IEEE* and João Eduardo Ferreira

Abstract—Over recent years, several sophisticated Process-Aware Information Systems (PAIS) have been proposed for managing business processes and automating large-scale scientific (e-Science) processes. Much of this success is due to their ability to provide generic functionality for modeling, execution and monitoring processes. These functionalities work well when process execution follows a well-behaved path towards achieving the models objectives. However, exceptions and anomalous situations that fall outside of the well-behaved execution path still pose a significant challenge to PAIS. The treatment for such exceptions usually involves interventions in systems by human operators, which result in significant additional cost for businesses. In this paper, we introduce a cost-aware recovery composition method that is able to find and follow recovery paths that reduce the cost of exception handling. From a practical point of view, our proposal reduces complexity and the need for manual interventions to handle exceptions. Finally, the feasibility of recovery mechanism is discussed from its implementation into WED-flow framework.

Index Terms—Process-Aware Information Systems, Reliability, Robustness, WED-flow, Exception Handling.

1 Introduction

Over recent years, several sophisticated Process-Aware Information Systems (PAIS) have been proposed [1], [2] for managing business processes and automating large-scale scientific (e-Science) processes. PAIS are software systems that manage and execute processes explicitly defined using process models and specifications [1]. These explicit specifications of process models enable the translation of business process and scientific application requirements into executable programs. While this strategy has worked well for specifying and executing the expected behaviors of processes, deviations and error recovery from the expected paths remain a significant challenge. For each expected path, there are many potential exceptions and failures that may arise, some expected and many unexpected. Consequently, providing robust execution (with appropriate handling of deviations and errors) is a complex and expensive task for current PAIS, since exception handling can become exponentially difficult when all possible outcomes and values are considered, and their treatment translated into procedural programs.

To a large extent, this structure of PAIS (favoring the expected over the exceptions) reflects the applications they model, which typically have a well-defined objective and a well-behaved path towards achieving that objective. Typically, this path can be explicitly specified and then translated into a sequence of state transitions for execution. In this paper, we will use a simplified car rental process (car reservation, credit card verification, car pick-up, and return) as an illustrative application for our approach. This money-making path (called the *primary path*) leads to a sequence of states that produce business profits at the end. However, primary paths also have many branches that lead to unsuccessful states. These branches are exceptions [3] that deviate from the primary path, for example, car rental process may include complications such as car return delays, accidents, and mechanical problems in cars. Trying to model these exceptions in a procedural implementation of PAIS is both non-trivial and expensive, since each state in the primary path may lead to several such exceptions.

The exception handling problem is both important and urgent. With the increasing importance of PAIS in modern business processes, more revenue-generating primary paths are added. Each primary path is accompanied by its own branches of exceptions and non-revenue outcomes that need to be handled properly. It is common for the exception handling of so many cases to grow much bigger than the primary paths in practical PAIS applications, leading to problems in maintenance, testing, and reuse. Furthermore, one of the main goals of PAIS is to support business process reengineering, which exacerbate the problem since the large body of exception handling code also needs to be reengineered. In practice, PAIS applications consist mainly of the primary paths, and the exceptions are handled by call center operators, who provide two kinds of services. For customers, they fix problems such as incomplete, duplicate, or cancelled purchases. For the service providers, they fix inconsistencies in business databases such as wrong charges or orders, and recovering from software or hardware system errors. The importance of the exception handling problem is demonstrated by the size of global contact center market, expected to reach USD 9.7 billion by 2019 [4].

To address the challenges of automated exception handling, the WED-flow (Workflow, Event processing and Dataflow) approach has been introduced [5], [6], [7] to provide a

- *A. L. Schwerz and R. Liberato are with the Department of Computing, at the Federal University of Technology - Paraná (UTFPR), Brazil E-mail: {andreluis, liberato}@utfpr.edu.br*
- *C. Pu is with the School of Computer Science, Georgia Institute of Technology, Atlanta, GA 30332, USA. E-mail: calton.pu@gatech.edu.*
- *J. E. Ferreira is with the Department of Computer Science, Institute of Mathematics and Statistics, at University of São Paulo, Brazil. E-mail: jef@ime.usp.br*

Manuscript received XXX XX, 2016; revised XXXXX XX, 2016.

1939-1374 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2824810, IEEE Transactions on Services Computing

method to specify exception handling procedures, the translation of the specification into executable programs, and the run-time environment to execute robust WED-flows that can recover automatically from errors and deviations from the primary path. WED-flow provides a dynamically composable environment in which several exception handling methods can be integrated into recovery paths that result in consistent outcomes. The main contribution of this paper is a system that builds on WED-flow mechanisms to provide an automated and optimized exception handling process, by calculating the cost and benefits of each recovery path, and choosing the recovery path with the best cost/benefits available. A practical demonstration system implements the cost-aware WED-flow system for the car rental application.

This paper is organized as follows. An overview of the WED-flow approach is described in Section 2. The WED-flow recovery strategies are discussed in Section 3. Section 4 presents cost-aware automated recovery of WED-flow. The experimental results are discussed in Section 5. Section 6 discusses related work. Finally, the conclusion is presented in Section 7.

2 THE WED-FLOW APPROACH

2.1 Summary of WED-flow concepts

The WED-flow (Workflow, Event processing and Data-flow) approach for modeling and implementation of business processes has been described previously [5], [6], [7]. We include a summary of main WED-flow concepts to make this paper self-contained.

Informally, a WED-flow model consists of a set of WED-states, WED-conditions, WED-transitions and WED-triggers. A *WED-state* is a set of attribute values (known as *WED-attributes*) that define a concrete workflow execution state. Specifically, WED-state includes sufficient data and execution states that will enable backward and forward recovery of a workflow. A *WED-condition* is a logical predicate defined over a specific set of WED-states. WED-condition captures events (changes of data states) and enables triggering of WED-transitions as part of workflow execution. A *WED-transition* is a step of WED-flow execution that transforms an input (WED-state) into an output (WED-state). Some WED-transitions have an inverse function (compensating step) named *WED-compensation*, where WED-compensation(WED-transition(input_WED-state)) = input_WED-state. Currently, we assume that the execution of WED-transitions (and WED-compensations) is encapsulated in atomic transactions such as SAGA steps [8]. A timeout for execution is specified for typical WED-transitions. The timeout expirations are abnormal behaviors that require exception handling. A *WED-trigger* binds a WED-condition to a WED-transition. When the WED-condition of a WED-trigger is satisfied, its associated WED-transition is executed.

A WED-flow instance is a specific occurrence or execution of a WED-flow process. An instance example for the car rental process occurs when a customer reserves a car. More concretely, the creation of a WED-flow instance occurs when a new WED-state (usually produced by an external event) satisfies the starting condition of WED-flow process. An instance finishes its execution when it reaches a WED-state that satisfies final condition of WED-flow process. Therefore, all WED-flow process has two special WED-conditions used for capturing initial and final WED-state. A WED-state may satisfy more than one WED-condition causing the divergence of a branch into two or more parallel branches. Each of these parallel branches produces data states by writing in a disjoint subset of wed-attributes. The convergence of two or more branches into a single subsequent branch occurs when a WED-state satisfies a WED-condition whose predicate is defined on WED-attributes of all subsets.

Multiple WED-flow instances are also supported. The WED-triggers evaluate WED-states from several instances. In other words, when a WED-state s is produced, each WED-trigger $g_j = \langle c_j, t_j \rangle$ must evaluate. If the associated WED-condition c_j is satisfied by s, then the associated WED-transition t_j is fired.

The execution history of a WED-flow is recorded as a sequence of WED-states, stored in a multiversion database [9] in an append-only fashion. An execution history h_i of WED-flow instance i starts from WED-state s_0, and contains $\langle$ WED-transition t_1, WED-state $s_1\rangle$, ... ,$\langle$WED-transition t_n, WED-state $s_n\rangle$, in which s_n is the final state of h_i. We omit the instance name h_i when the context is clear.

2.2 Business Process Modeling in WED-flow

The abstract model summarized in Section 2.1 is implemented by concrete software tools that execute WED-flow instances providing appropriate consistency properties. The specification of a business process as an instance of WED-flow model and its execution is outlined in this section. Further details of WED-flow application specification and execution can be found in [6].

The WED-flow development methodology models a business process as a set of activities that transform the system and data state (WED-states). Activities are linked together by explicit events, defined as predicates describing changes on WED-states. The modeling of a business activity starts from the primary path, the important activities and related events that generate revenues, or achieve some other important business goals. Using WED-flow, the primary path starts from an initial WED-state, which serves an input to a WED-transition, generating an output WED-state. The execution of a WED-flow instance (implemented by WED-triggers) follows the sequence of events captured by WED-state changes to a final WED-state.

An implementation of WED-flow starts from the definition of WED-attributes, the set of schemas that describe the databases that store WED-states (including data and WED-flow execution states). Business activities are specified as functional mappings (WED-transitions), which transform input WED-states into output WED-states (atomically). The execution of a WED-transition is initiated by WED-triggers (pairs of WED-conditions and WED-transitions), which are conceptually similar to continual queries [10]. Through the WED-condition, a WED-trigger ensures that the preconditions of its WED-transition (including the availability of appropriate input) are satisfied before activating the WED-transition. WED-transitions are protected by transactional boundaries for maintaining system data consistency, and they are similar to SAGA steps [8]. The successful execution of a WED-transition takes the WED-flow system to

1939-1374 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2824810, IEEE Transactions on Services Computing

an output WED-state, which (under normal circumstances) will trigger the next step in the primary path by satisfying the WED-condition in the next WED-trigger.

We have developed effective software tools [11] that implement the WED-flow execution outlined above, including the support for WED-state database schema (WED-attributes), WED-conditions (changes of WED-states), and WED-transitions (activities that take a consistent input WED-state into a consistent output WED-state). Practical applications built on WED-flow include DECA online business license management system [12] for the São Paulo State Government and CEGH genetic testing system [13] for the Human Genome Research Center in Brazil. In the next subsection, a concrete example shows how the primary path of the business process is described using the WED-flow approach.

2.3 Primary Path Example: Car Rental

We will use a simplified online car rental process to illustrate the advantages of business process modeling and implementation using WED-flow approach. The car rental process manages rental orders, payment method verification, car pickup, and return locations. The illustrative primary path starts from the placement of an order for car rental and concludes with the car return and payment. The main steps of the car rental primary path are outlined in Fig. 1(a), where each square represents an activity, which is specified by a corresponding WED-transition in Fig. 1(b). In the graphical notation of Fig. 1(b), a circle indicates a WED-condition and a pentagon represents a WED-transition. In addition, a pair (circle and pentagon) represents a WED-trigger and the rectangles denote the externally visible WED-states during the execution of primary path.

The WED-attributes of the car rental process WED-state database are divided into three groups (Fig. 1(c)): `WED-Order`, `WED-Customer`, and `WED-Vehicle`, which may be implemented as individual tables. We note that these tables are quite different from the classic database tables for (all) orders, (all) customers, and (all) vehicles. The rows in Fig. 1(c) contain the concrete WED-states recorded during the execution of the primary path. All the rows refer to the fulfillment of the same order (`ID` $=$ `1`), which was placed by a single customer (`ID` $=$ `C4`), and in this example, the same car (`ID` $=$ `V1`).

The primary path of the car rental process starts from a customer login and initiating the request for renting a car. The request is created by an initial WED-transition that inserts the initial WED-state $\mathtt{S}_0$ into the WED-state database (top row denoted by $\mathtt{S}_0$ in Fig. 1(c)):

$$\mathtt{S}_0 = \langle\ \mathtt{1}, \mathtt{NYC}, \mathtt{01\text{-}01}, \mathtt{01\text{-}03}, [\mathtt{C}], \mathtt{Requested}, \mathtt{C4}, \mathtt{Not\ Validated}, \mathtt{null}, \mathtt{null}, \mathtt{null}\ \rangle$$

The WED-state $\mathtt{S}_0$ triggers WED-condition c_1, which activates activity a_1: *Customer Login* (WED-transition t_1 in Fig. 1(b)). In t_1, the validation of customer creates WED-state $\mathtt{S}_1$, which satisfies WED-condition c_2 and triggers activity a_2: *Vehicle Assignment* (WED-transition t_2). A successful assignment of a compact vehicle in a_2 creates WED-state $\mathtt{S}_2$. The rows in Fig. 1(c) show the successful execution of the activities in primary path from a_1 through a_6. Concretely, the activities a_3: *Payment Method Validation*, a_4: *Car Pick-up*, a_5: *Car Return*, and a_6: *Send Invoice*, are triggered by their respective WED-conditions when WED-states $\mathtt{S}_2$, $\mathtt{S}_3$, $\mathtt{S}_4$, and $\mathtt{S}_5$ are created. The successful conclusion of the primary path is the (paying) WED-state $\mathtt{S}_6$ in Fig. 1(b).

For simplicity of presentation, the car rental example in Fig. 1 describes a sequential primary path, where each WED-state only satisfies one WED-condition, which activates only one WED-transition. Readers familiar with event processing may expect high parallelism from the event-based WED-flow specification, capable of describing the most complex control-flow behaviors proposed in [14], [15]. Intuitively, all WED-conditions are evaluated in parallel each time a new WED-state is created, which can activate many parallel WED-transitions if appropriate. A full discussion of the WED-flow specification expressiveness is beyond the scope of this paper.

An important advantage of the WED-flow approach is the preservation of correctness properties and system consistency throughout the execution of a WED-flow instance that is discussed in the following subsection.

2.4 Consistency Properties Preserved by WED-flow

First, we will introduce some notation to describe the preservation of correctness properties and system consistency throughout the execution of a WED-flow instance (e.g., the primary path). These consistency properties are derived from the WED-flow definitions:

Consistency of successful path. Let m-tuple $\langle s_1, s_2, ..., s_m \rangle$ be a successful sequence of WED-states produced by a WED-flow instance in which no inconsistent data is generated. In WED-flow execution, a process instance leads the application from one AWIC-consistent state to another AWIC-consistent state. Application-Wide Integrity Constraints (AWICs) are described as predicates (WED-conditions) over the WED-attributes of process and define integrity constraints that cover the databases of application. AWICs also include the semantic rules that express business rules (e.g., total number of available cars). Throughout its execution, a process instance may create several intermediate WED-states (e.g., s_1 to s_{m-1}), which are referred to as transaction-consistent WED-states.

Consistency of recovery paths. When the execution of an instance deviates from the primary path (e.g., timeout expiration for WED-transition), an inconsistent WED-state may be created. The automated WED-flow recovery proceeds at two levels. At the syntactic level, active transactions are aborted and their (partial) updates are undone. At the semantic level, system data consistency may be recovered in two ways: (1) backward recovery; or (2) forward recovery.

To illustrate the advantages of the recovery techniques described here, in next subsection, two concrete exception examples from the car rental order are discussed.

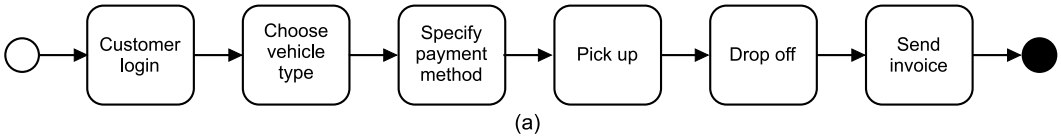
3 Building Blocks of WED-flow Recovery

3.1 Modular Recovery Components (Backward and Forward)

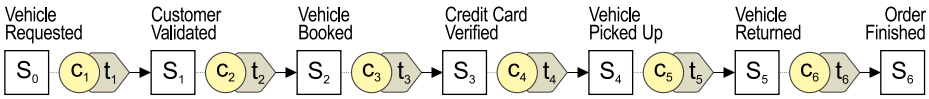
A recovery path is a sequence of recovery steps that takes the application database from an (potentially) inconsistent

1939-1374 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2824810, IEEE Transactions on Services Computing



(a)



(b)

	WED-Order						WED-Customer		WED-Vehicle		
	ID	City	Pick-up Date	Drop-off Date	Preference	Status	ID	Status	ID	Category	Status
S_0	1	NYC	01-01	01-03	[C]	Requested	C4	Not Validated			
S_1	1	NYC	01-01	01-03	[C]	Validated	C4	Validated			
S_2	1	NYC	01-01	01-03	[C]	Validated	C4	Validated	V1	Compact	Booked
S_3	1	NYC	01-01	01-03	[C]	CC Verified	C4	Validated	V1	Compact	Booked
S_4	1	NYC	01-01	01-03	[C]	CC Verified	C4	Validated	V1	Compact	Picked-up
S_5	1	NYC	01-01	01-03	[C]	CC Verified	C4	Validated	V1	Compact	Returned
S_6	1	NYC	01-01	01-03	[C]	Finished	C4	Validated	V1	Compact	Returned

(c)

Fig. 1. Car Rental Primary Path: (a) main component activities in car rental; (b) WED-flow model of the primary path; and (c) concrete WED-states created by the execution of the primary path.

state back to an AWIC-consistent state. The recovery steps consist of two special kinds of WED-transitions:

- Undo recovery steps that perform backward recovery. They take a WED-flow instance to a previous AWIC-consistent state (e.g., in the execution of primary path).
- Redo recovery steps that perform forward recovery. They take a WED-flow instance to a new AWIC-consistent state that may not have been executed (e.g., "further down" in the primary path).

The undo and redo steps are designed and implemented at the same time as the WED-transitions in the primary path. The undo recovery steps consist of WED-compensation steps, one for each WED-transition. The purpose of WED-compensation steps is to "roll back" WED-states so the WED-flow instance can restore AWIC-consistency caused by failures. The redo recovery steps are components of secondary paths to the primary path. The purpose of redo recovery steps is to "roll forward" WED-states so the WED-flow instance can resume, preferably somewhere further along the primary path that is unaffected by the failure that interrupted the original primary path execution.

Although the design and implementation of undo and redo steps are application-dependent, they are typically closely related to the WED-transitions in the primary path. In the next subsection, we use the car rental example to illustrate the design strategy for undo and redo steps. More details about recovery strategies are described in [7].

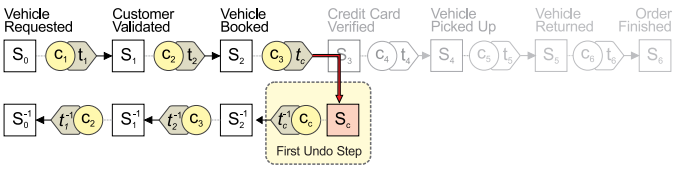
3.2 Cost-Insensitive Recovery Examples

First, we use backward recovery to handle the cancellation of car reservation process. A cancellation may occur for several reasons, e.g., explicit customer request for cancellation, or failure to pick up car at specified time (detected by timeout). Fig. 2(a) illustrates the backward recovery of car cancellation. The cancellation event (either by request or timeout) is detected when WED-state $\mathtt{S}_c$ is created by the cancelling activity or timeout handler. Since the semantics of cancellation prevent further execution, backward recovery path starting from the shaded undo step is followed, taking the WED-state $\mathtt{S}_c$ as input and producing the WED-state $\mathtt{S}_2^{-1}$ as output. The WED-state $\mathtt{S}_2^{-1}$ is equivalent to WED-state $\mathtt{S}_2$ in the primary path, which is transaction-consistent, but not AWIC-consistent. Due to the absence of side-effects in the primary path before vehicle assignment, the backward recovery path is able to continue following a sequence of WED-compensations t_2^{-1} and t_1^{-1}, (additional undo steps) to compensate for WED-transitions t_2 and t_1, and reaching the AWIC-consistent WED-state $\mathtt{S}_0^{-1}$. These WED-compensations are carefully designed and implemented to produce the WED-states equivalent to WED-states used as input in the execution of WED-transitions. For example, in illustrative example of Fig. 2, the WED-compensation t_2^{-1} is executed to "unassign" the compact vehicle in the car rental process. After its execution, the compact vehicle is again available for a new reservation. The WED-state $\mathtt{S}_1^{-1}$ generated by t_1^{-1} is semantically equivalent to the WED-state $\mathtt{S}_1$. Informally, two WED-states are equivalent if both satisfy the same WED-conditions. Finally, the WED-compensation t_1^{-1} is executed, reaching the AWIC-consistent WED-state $\mathtt{S}_0^{-1}$.

Complementing the previous backward recovery example, we use forward recovery to include a new credit card. Fig. 3(a) illustrates a deviation from the primary path when a customer has expired credit card: WED-transition t_3 is outside of primary path, producing a WED-state $\mathtt{S}_d$. The forward recovery path is activated by $\mathtt{S}_d$ with the redo step,

1939-1374 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2824810, IEEE Transactions on Services Computing



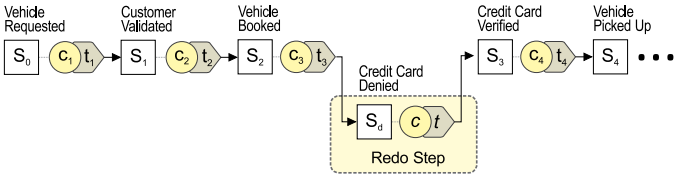
(a)

	WED-Order						WED-Customer		WED-Vehicle		
	ID	City	Pick-up Date	Drop-off Date	Preference	Status	ID	Status	ID	Category	Status
S_0	1	NYC	01-01	01-03	[C]	Requested	C4	Not Validated			
S_1	1	NYC	01-01	01-03	[C]	Validated	C4	Validated			
S_2	1	NYC	01-01	01-03	[C]	Validated	C4	Validated	V1	Compact	Booked
S_c	1	NYC	01-01	01-03	[C]	To cancel	C4	Validated	V1	Compact	Booked
S_2^{-1}	1	NYC	01-01	01-03	[C]	Validated	C4	Validated	V1	Compact	Booked
S_1^{-1}	1	NYC	01-01	01-03	[C]	Validated	C4	Validated	V1	Compact	Canceled
S_0^{-1}	1	NYC	01-01	01-03	[C]	Canceled	C4	Validated	V1	Compact	Canceled

(b)

Fig. 2. Illustrative backward recovery example for car cancellation: (a) WED-flow model of primary path and backward recovery path (b) WED-states produced by the entire WED-flow including backward recovery.

which enables the customer to include a new credit card, generating a new transaction-consistent WED-state S_3. At this point, the WED-flow instance returns again to primary path as illustrated in Fig. 3. If the customer does not inform a new credit card, the timeout handler notifies a cancellation event (e.g., S_c in Fig. 2). Both cases are naturally supported by WED-flow.



(a)

	WED-Order						WED-Customer		WED-Vehicle		
	ID	City	Pick-up Date	Drop-off Date	Preference	Status	ID	Status	ID	Category	Status
S_0	1	NYC	01/01	03/01	[C]	Requested	C4	Not Validated			
S_1	1	NYC	01/01	03/01	[C]	Validated	C4	Validated			
S_2	1	NYC	01/01	03/01	[C]	Validated	C4	Validated	V1	Compact	Booked
S_d	1	NYC	01/01	03/01	[C]	CC Denied	C4	Validated	V1	Compact	Booked
S_3	1	NYC	01/01	03/01	[C]	CC Verified	C4	Validated	V1	Compact	Booked
S_4	1	NYC	01/01	03/01	[C]	CC Verified	C4	Validated	V1	Compact	Picked-up
S_5	1	NYC	01/01	03/01	[C]	CC Verified	C4	Validated	V1	Compact	Returned
S_6	1	NYC	01/01	03/01	[C]	Finished	C4	Validated	V1	Compact	Returned

(b)

Fig. 3. Illustrative forward recovery example for handling a denied payment: (a) WED-flow model of primary path and forward recovery path (b) WED-states produced by the WED-flow including forward recovery.

Forward recovery paths can be combined with backward recovery paths. This flexible composition at run-time introduces an interesting challenge, namely, the problem of choosing the best secondary path from the various possible combinations. This is particularly the case when different secondary paths have different costs and benefits that may vary dynamically. The cost-aware recovery composition method described in the following section is designed to find the secondary path that maximizes the profit (or minimizes the cost) of the current WED-flow instance despite failures.

3.3 Cost Considerations in the Recovery of AWIC-Consistency

The scenarios outlined in the previous section (backward recovery and forward recovery) illustrate the WED-flow approach to restore AWIC-consistency when a failure occurs. However, the restoration of AWIC-consistency by itself may or may not achieve the business objective of adopting a recovery path while minimizing losses (by maximizing remaining profits) when a failure occurs. Let us consider the various alternatives that can help a rental car company recover from a car breakdown.

The car rental primary path described in Fig. 1 generates the maximum profits when the execution reaches a successful ending. However, the primary path also has several potential branches for each activity to cover the less successful situations (secondary paths) such as failures. We will use a specific car reassignment scenario to illustrate the several possible secondary paths enabled by WED-flow. Let us consider the following situation:

- All compact vehicles have been rented, or assigned to upcoming rentals.
- A compact vehicle with a serious mechanical problem has been identified.
- At least one of the upcoming rental assignments needs to be changed to accommodate the existing (approved) request that no longer can be fulfilled by the car that just broke.

Note that the broken car is an external event that introduces an inconsistency into the application, i.e., the invariant of total number of available cars has been reduced by one. It is non-trivial to recover from this type of AWIC-consistency violation, since there is no internal record of how the external event originated. Traditionally, recovery from such events is left to human operators (e.g., in call centers) by executing a decision tree defined by business process designers. In the broken car scenario, the easiest solution is to replace the broken with another similar car. In case of sold-out, the goal is selecting a suitable replacement vehicle that will satisfy customers preferences. The candidate secondary paths include: (1) find an available vehicle from higher categories, (2) transfer a suitable vehicle from other locations, (3) give a discount coupon and Uber credits to end the car rental, plus other potential business solutions. Currently, call center operators must determine which of these alternatives are feasible, and offer them to customers. Unfortunately, the relative costs of various alternatives are dynamically variable and typically unavailable to the operators. Consequently, it is difficult for a PAIS-based business process to find the secondary path that optimizes the cost/benefits.

In the following section, we will describe the WED-flow building block to calculate profits and compose (automatically) feasible secondary paths with their costs, so we can choose the secondary path with highest profit.

3.4 WED-flow Support for Estimated Profit Calculations

The design and implementation of WED-flow facilitates the introduction of "new" features that can be difficult to handle

1939-1374 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2824810, IEEE Transactions on Services Computing

in traditional PAIS. The support for cost and benefit calculations is a good example of such features. For simplicity, we use the term *profit* to denote the difference between all costs and revenues produced by an execution path. In procedural programs, the tracking and calculation of costs and revenues of each recovery path is difficult. The main problem is that both the costs and benefits may vary dynamically at runtime due to the context of execution, dependencies among the workflow components, and often the current system state. Consequently, the tracking and calculation of profit is as difficult as the exception handling itself, subject to combinatorial explosion when each recovery path is described explicitly in procedures.

In contrast to procedural PAIS, in WED-flow recovery paths are composed dynamically and each component is triggered by an event, implemented as appropriate WED-state changes. Since WED-states contain all the relevant information about the WED-flow instance being executed, we calculate the profit of each execution path (including both the primary path and the secondary paths) from its WED-states. We also augment the WED-state with a new WED-attribute, called `C/R` for the estimated cost and revenue of activities contained in an execution path.

Definition 1. *CR.* Consider a WED-flow instance Wf_i consisting of WED-transitions $\langle t_1, \ldots, t_m \rangle$. Let $h_i = \langle s_0, s_1, \ldots, s_m \rangle$ be an execution history of Wf_i, formed by a sequence of WED-states produced by the corresponding WED-transitions in Wf_i. For each s_j, we calculate the value $v_{cr}(s_j)$ that represents the value (cost or revenue) achieved by t_j in the execution history h_i. The CR function of h_i is given by:

$$CR(h_i) = \sum_{j=1}^{m} v_{cr}(s_j)$$

The interpretation of profit calculation of CR function above is straightforward when h_i starts from an initial WED-state and ends in a final WED-state, e.g., from S_0 to S_6 in Fig. 1.

We generalize Definition 1 on concrete execution history h_i to enable the calculation of an estimated profit of an incompletely executed history $h_{inc} = \langle s_0, s_1, \ldots, s_r \rangle$, where $r < m$. The calculation of such estimated profits proceeds by simulating the execution of WED-transitions $\langle t_{r+1}, \ldots, t_n \rangle$, $n \leq m$, by continuing the execution of Wf_i from WED-state s_r. The simulated execution (with hypothetical but reasonable input for each WED-transition, and without causing any side effects) of various possible outcomes of Wf_i allows us to estimate its achievable profits. The estimated profits enable a cost-conscious comparison that results in a choice of a path with maximum profits. We assume that there are k feasible continued executions (enumerated from f_1 through f_k) to the end of Wf_i. We define the expected profits of Wf_i with history h_i, when continuing from the current WED-state (s_r) as follows.

Definition 2. *ECR.* The expected profits of Wf_i with WED-transitions $\langle t_1, \ldots, t_m \rangle$, but in the middle of an incomplete execution $h_{inc} = \langle s_0, s_1, \ldots, s_r \rangle$, $r < m$, is the sum of profits achieved by the ongoing execution $CR(h_{inc})$ and the maximum of estimated profits from feasible hypothetical paths that Wf_i can take from the current WED-state s_r. The choice of hypothetical paths $\{f_1, \ldots, f_k\}$ is restricted by the feasibility of estimating the input values for their WED-states. The process of finding reasonable input values for a hypothetical path will be further explained below.

$$ECR(Wf_i, h_r) = CR(\langle s_1, \ldots, s_r \rangle) + \max_j \{CR(f_j = \langle s_{r+1}, \ldots, s_{m(j)} \rangle), 1 \leq j \leq k\}$$

We denote the ending WED-state of a hypothetical execution f_j by $s_{m(j)}$, since each hypothetical execution may end at any WED-state for the purpose of ECR calculation. Readers familiar with optimization problem may notice that such a general definition of the ECR function allows a potentially large number of hypothetical paths, which may introduce a computational problem for finding their maximum, since optimization problems are often NP-hard. Since the focus of this paper is on automated and cost-conscious recovery methods in WED-flow, we simplify the calculation of ECR to consider only feasible paths f_j of length 1, i.e., $s_{m(j)} = s_{r+1}$. This is a greedy algorithm [16] and we will use the acronym $GECR$ (for Greedy ECR) to denote the optimized result from the greedy algorithm.

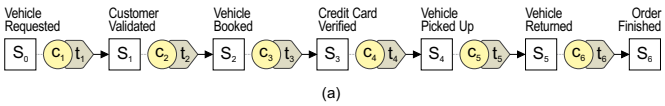
The greedy algorithm works well when activities that are semantically dependent (complex interdependence) are encapsulated in WED-transitions, and the optimization space of each WED-transition is convex [17]. In this case, the sum of local maximum profits (from each WED-transition) becomes the global maximum profit. This assumption is valid in the car rental example, where each WED-transition in the primary path is independent of others. This assumption is valid in many practical applications. In addition, we are exploring promising ideas such as the systematic removal of inter-component dependencies from workflows by merging the components with dependencies into a composite component. A full treatment of the general case, where dependencies may arise among the WED-transitions, is subject of ongoing research and beyond the scope of this paper. In additon, probabilistic models for estimating profitable paths that cannot be converted into parameters that represent costs and revenues also are beyond the scope of this work.

We now apply the greedy algorithm to calculate the estimate profits of the primary path. Fig. 4 includes the profits as a new column in the WED-state (the `C/R` field), where the profits are calculated by following the primary path and choosing the lowest cost and highest revenue option for the next WED-transition when multiple choices are feasible. A good example is WED-transition t_2 that has as input a WED-state in which customers preferences may result in multiple acceptable alternatives of vehicles for reservation. This WED-transition is designed and implemented for choosing the highest revenue option among available vehicles that satisfy the customer preferences. More details on the different design methods of process steps and their implications are beyond the scope of this paper. In Fig. 4, the execution of WED-transition t_2 (vehicle assignment) adds an estimated rental income ($20) according to the vehicle category that the customer booked, and WED-transition t_5 (car return) includes the estimated depreciation of vehicle

1939-1374 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2824810, IEEE Transactions on Services Computing

during the rental period and vehicle cleaning costs when it is returned (-$3). For this car rental primary path (h_i) example of a compact car, the GECR(Wf_i ,h_i) adds up to $17.



(a)

	WED-Order						WED-Customer		WED-Vehicle			
	ID	City	Pick-up Date	Drop-off Date	Preference	Status	ID	Status	ID	Category	Status	C/R
S_0	1	NYC	01-01	01-03	[C]	Requested	C4	Not Validated				
S_1	1	NYC	01-01	01-03	[C]	Validated	C4	Validated				
S_2	1	NYC	01-01	01-03	[C]	Validated	C4	Validated	V1	Compact	Booked	$20
S_3	1	NYC	01-01	01-03	[C]	CC Verified	C4	Validated	V1	Compact	Booked	
S_4	1	NYC	01-01	01-03	[C]	CC Verified	C4	Validated	V1	Compact	Picked-up	
S_5	1	NYC	01-01	01-03	[C]	CC Verified	C4	Validated	V1	Compact	Returned	-$3
S_6	1	NYC	01-01	01-03	[C]	Finished	C4	Validated	V1	Compact	Returned	

(b)

Fig. 4. Car rental primary path including the cost and revenue.

The primary path example in Fig. 4 has one single path $\langle S_0, S_1, \ldots, S_6 \rangle$ when successfully executed to the end. In the following sections, we will describe the use of GECR in the automated cost-conscious recovery of WED-flows when their executions may deviate from the primary path, by automated combination of (pre-defined) backward and forward recovery steps.

4 Cost-aware Automated Recovery of WED-flow

4.1 Automated and Cost-conscious Recovery in WED-flow

The automated recovery from deviations from primary path is handled by WED-flow through a dynamic composition of two kinds of consistency restoration procedures that include the undo and redo steps described in Section 3.1. The dynamic composition process is divided into two parts: autogen-undo and autogen-redo, to choose and compose the appropriate undo and redo recovery steps. In Fig. 5, we use a concrete recovery example (handling the impact of a mechanical failure of car that was assigned to a reservation) to illustrate the dynamic composition process by a combination of autogen-undo and autogen-redo.

When a car becomes unavailable due to mechanical failure, the event is handled by the WED-flow recovery process in three stages: (1) determination of situation, (2) autogen-undo, and (3) autogen-redo. In the first stage, a search for any references to the broken car is performed in the WED-states of the WED-flow instance execution. If no references are found (car is unassigned), then the car can be sent to repairs without further action. In this example, the car is found to be assigned to a rental process, which is in WED-state S_3 awaiting pickup. Recovery is needed since the pickup will fail due to the broken car becoming unavailable.

The recovery process moves to second stage by invoking autogen-undo to "unassign" the broken vehicle. This is achieved by compensating for the WED-states affected by the failure event. In the example, it starts from S_3 and continues until reaching the goal of "unassign" by undoing the WED-state in which the car was assigned (S_2). To undo the assignment made by S_2, the autogen-undo procedure generates the WED-compensations up to WED-state S_1, called the *stop point* for the automated recovery method.

The autogen-undo procedure starts from the current WED-state (S_3), called S_3^{-1} to avoid confusion with the forward execution of the primary path. We assume that at the time of implementation of each WED-transition t_i, a corresponding WED-compensation t_i^{-1} has been also implemented, where semantically $(t_i \circ t_i^{-1}) = I$. The autogen-undo procedure follows the sequence of WED-states starting from the current (S_3) by invoking WED-compensation t_3^{-1} with S_3^{-1} as input. Successful execution of t_3^{-1} generates S_2^{-1}, which becomes the input to WED-compensation t_2^{-1}, since the previous WED-transition was t_2^{-1}. The undo recovery process continues until the autogen-undo procedure reaches the stop point (S_1), illustrated in Fig. 5.

The general algorithm for autogen-undo can be described as follows:

1) Obtain the current WED-state s_c at the time of failure, the last entry in the WED-state database and the starting point of the recovery path;
2) Find the stop point WED-state s_{sp}, the goal of the recovery path; this search starts from the failure that caused one of the fields in s_c to become inconsistent. (In Fig. 5 it is the broken car that caused the car assignment to become invalid). The WED-state that served as input to the WED-transition that generated the inconsistent field (car assignment) is the stop point;
3) The undo recovery steps thus generated should have appropriate WED-conditions that will trigger successive undo WED-transitions, starting from s_c^{-1} until reaching s_{sp}.

The autogen-undo restores the WED-flow to a consistent state by compensating for the intermediate inconsistent states. The third step of the recovery process attempts to find an alternative solution to profitability, e.g., by returning to the primary path. The third stage is implemented by autogen-redo, which starts from the stop point WED-state s_{sp} (S_1^{-1} in the car rental example). To avoid confusion with backward recovery, the stop point WED-state s_{sp} used as input for autogen-redo is illustrated as $S_1^{-1^{+1}}$ in Fig. 5. This WED-state is then evaluated for all WED-conditions to determine which WED-transitions (redo steps) have to be executed. In the car rental example, the WED-transition t_2 is triggered, and its execution produces the WED-state $S_2^{-1^{+1}}$. In other words, the execution of t_2 assigns a new vehicle for this reservation order according to the available vehicles and the customer's preferences. Next, the WED-state $S_2^{-1^{+1}}$ is also evaluated by all WED-conditions, which determines the execution of WED-transition t_3. This execution produces the WED-state $S_3^{-1^{+1}}$. The autogen-redo procedure ends when a step produces a state equivalent to the current state s_c before starting the autogen-undo, e.g., by returning to the primary path. However, when the primary path is not achieved, the autogen-redo procedure may also be finalized when an AWIC-consistent WED-state is reached. A concrete secondary path that illustrates this situation is described in the following section.

1939-1374 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2824810, IEEE Transactions on Services Computing

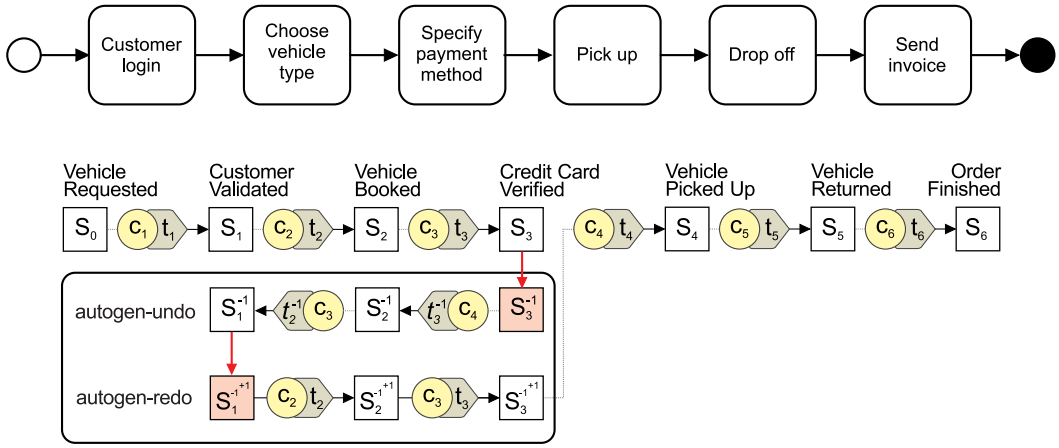


Fig. 5. Automatic recovery in WED-flow

The general algorithm for autogen-redo can be described as follows:

1) Obtain the WED-state equivalent to s_{sp} produced through the autogen-undo procedure;
2) The redo recovery steps thus generated should have appropriate WED-conditions that will trigger successive redo WED-transitions, starting from s_{sp} (last WED-state executed by autogen-undo) until reaching a WED-state equivalent to WED-state s_c.

The autogen-redo procedure ended because the WED-state $S_3^{-1^{+1}}$ is equivalent to the current state S_3 at the time of the inconsistency is detected. After WED-state $S_3^{-1^{+1}}$ has been created, the process instance can go ahead with its normal execution, so as to reach the activities of *car pick-up*, *car return* and *send invoice*, as illustrated in Fig. 5.

The autogen-undo and autogen-redo procedures produce a secondary path that does not involve the compact vehicle that was found to have a mechanical failure. There are several secondary paths, which depend on a combination of factors concerning process instance requirements, available resources and recovery costs. In the next section, some possible secondary paths obtained from execution of autogen-undo and autogen-redo procedures are described.

4.2 Secondary Path Examples from Recovery

For concreteness, we outline some examples of secondary paths that illustrate the car rental scenario. In these examples, all WED-states are numbered sequentially. However, the lines in the execution history that are highlighted refer to the additional steps taken by the autogen-undo and autogen-redo. In addition, the costs and revenues at each state transition are presented.

Fig. 6 illustrates a secondary path, in which the compact vehicle that was previously reserved has to be canceled and another vehicle is automatically obtained through the combination of autogen-undo and autogen-redo procedures. Firstly, the WED-states S_1, S_2, and S_3 indicate conclusion of the activities of *customer validation*, *vehicle assignment* and *payment method validation*, respectively. Next, the recovery composition method generates the WED-conditions and WED-transitions that will carry out the undo of the WED-states that need to be canceled. The autogen-redo proceeds with WED-states S_4 and S_5 that compensate for the vehicle assignment and payment method validation activities. Then the autogen-redo process is able to create the WED-states S_6 and S_7 by re-executing WED-transitions t_2 and t_3. The recovery process ends with WED-state S_7, which corresponds to the WED-state at the beginning of the cancellation (i.e. S_3). From WED-state S_7, onwards, the WED-flow continues its normal execution, to reach the WED-states S_8, S_9 and S_{10}.

	WED-Order						WED-Customer		WED-Vehicle			
	ID	City	Pick-up Date	Drop-off Date	Preference	Status	ID	Status	ID	Category	Status	C/R
S_0	1	NYC	01-04	01-05	[E, C]	Requested	C4	Not Validated				
S_1	1	NYC	01-04	01-05	[E, C]	Validated	C4	Validated				$20
S_2	1	NYC	01-04	01-05	[E, C]	Validated	C4	Validated	V1	Compact	Booked	
S_3	1	NYC	01-04	01-05	[E, C]	CC Verified	C4	Validated	V1	Compact	Booked	
S_4	1	NYC	01-04	01-05	[E, C]	Unverified	C4	Validated	V1	Compact	Booked	
S_5	1	NYC	01-04	01-05	[E, C]	Unverified	C4	Validated	V1	Compact	Canceled	-$20
S_6	1	NYC	01-04	01-05	[E, C]	Unverified	C4	Validated	V3	Economy	Booked	$15
S_7	1	NYC	01-04	01-05	[E, C]	CC Verified	C4	Validated	V3	Economy	Booked	
S_8	1	NYC	01-04	01-05	[E, C]	CC Verified	C4	Validated	V3	Economy	Picked-up	
S_9	1	NYC	01-04	01-05	[E, C]	CC Verified	C4	Validated	V3	Economy	Returned	
S_{10}	1	NYC	01-04	01-05	[E, C]	Finished	C4	Validated	V3	Economy	Returned	

Fig. 6. Secondary path: another vehicle within the customers preferences is obtained

Although the replacement vehicle is different from the broken vehicle originally reserved, the autogen-redo procedure was able to find one that meets the customers preferences. In this example, an economy vehicle (of higher value) replaces the compact vehicle without additional cost to the customer. Note that the costs and revenues at each WED-transition are explicitly represented in the WED-state database (right side of Fig. 6). For example, the process instance has accumulated revenue of $20 at the time when WED-state S_3 is written. As soon as the compact vehicle is canceled, a financial loss is calculated by the autogen-undo procedure. The autogen-redo procedure re-executes WED-transitions t_2 and t_3 and a new CR function is calculated.

1939-1374 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2824810, IEEE Transactions on Services Computing

The instance has now achieved a CR function of $15 instead of the $20 that was obtained before the cancellation.

Fig. 7 shows a similar scenario in which a higher-category vehicle is selected to replace the canceled vehicle. In this example, the customer has reserved a premium vehicle that has been replaced by a luxury vehicle. We take into consideration that a luxury vehicle is a compatible replacement for the premium vehicle requested by the customer. Even though this alternative may not be the normal recovery procedure for car rental companies, it may result in some profit, as illustrated in Fig. 7. The secondary path has a CR function of $50 instead of the $90 that was obtained before the cancellation.

	WED-Order						WED-Customer		WED-Vehicle			
	ID	City	Pick-up Date	Drop-off Date	Preference	Status	ID	Status	ID	Category	Status	C/R
S_0	1	NYC	01-04	01-05	[P]	Requested	C4	Not Validated				
S_1	1	NYC	01-04	01-05	[P]	Validated	C4	Validated				
S_2	1	NYC	01-04	01-05	[P]	Validated	C4	Validated	V4	Premium	Booked	$90
S_3	1	NYC	01-04	01-05	[P]	CC Verified	C4	Validated	V4	Premium	Booked	
S_4	1	NYC	01-04	01-05	[P]	Unverified	C4	Validated	V4	Premium	Booked	
S_5	1	NYC	01-04	01-05	[P]	Unverified	C4	Validated	V4	Premium	Canceled	-$90
S_6	1	NYC	01-04	01-05	[P]	Unverified	C4	Validated	V5	Luxury	Booked	$50
S_7	1	NYC	01-04	01-05	[P]	CC Verified	C4	Validated	V5	Luxury	Booked	
S_8	1	NYC	01-04	01-05	[P]	CC Verified	C4	Validated	V5	Luxury	Picked-up	
S_9	1	NYC	01-04	01-05	[P]	CC Verified	C4	Validated	V5	Luxury	Returned	
S_{10}	1	NYC	01-04	01-05	[P]	Finished	C4	Validated	V5	Luxury	Returned	

Fig. 7. Secondary path: a higher-category vehicle is selected

Figures 6 and 7 depict recovery scenarios, in which secondary paths that satisfy the customers preferences have been found. Unlike the previous examples, Fig. 8 illustrates an example, in which no vehicle that satisfies the customers preferences is found. More specifically, the autogen-redo procedure terminates early when WED-transition t_2 produces the WED-state S_8. Termination before achieving a WED-state equivalent to the current state at the time of the cancellation is a special case of the autogen-redo procedure. The WED-flow approach allows multiple termination states for a process [7]. In this example illustrated by Fig. 8, the termination state S_8 for the process instance is an AWIC-consistent state, from the WED-flow perspective. From the business perspective, this termination is an unprofitable way of handling cancellations; it also lowers customer satisfaction for failing to address customer needs. By automating the recovery process, WED-flow improves both profitability and customer satisfaction.

	WED-Order						WED-Customer		WED-Vehicle			
	ID	City	Pick-up Date	Drop-off Date	Preference	Status	ID	Status	ID	Category	Status	C/R
S_0	1	NYC	01-04	01-05	[C]	Requested	C4	Not Validated				
S_1	1	NYC	01-04	01-05	[C]	Validated	C4	Validated				
S_2	1	NYC	01-04	01-05	[C]	Validated	C4	Validated	V1	Compact	Booked	$20
S_3	1	NYC	01-04	01-05	[C]	CC Verified	C4	Validated	V1	Compact	Booked	
S_4	1	NYC	01-04	01-05	[C]	Unverified	C4	Validated	V1	Compact	Booked	-$20
S_5	1	NYC	01-04	01-05	[C]	Unverified	C4	Validated	V1	Compact	Canceled	
S_6	1	NYC	01-04	01-05	[C]	Unverified	C4	Validated	V1	Compact	Not Found	

Fig. 8. Secondary path: nothing can be done and the customers reservation is canceled

Fig. 9 depicts an example in which the customer receives a bonus because his reservation was canceled. This bonus refers to some type of credit that can be used in future reservations or in some partner company (e.g., Uber). This strategy is commonly used by several companies in case of cancellations. From the customer's point of view, a bonus may minimize his dissatisfaction, but it also increases the costs of the company to handle exceptions.

	WED-Order						WED-Customer		WED-Vehicle			
	ID	City	Pick-up Date	Drop-off Date	Preference	Status	ID	Status	ID	Category	Status	C/R
S_0	1	NYC	01-04	01-05	[C]	Requested	C4	Not Validated				
S_1	1	NYC	01-04	01-05	[C]	Validated	C4	Validated				
S_2	1	NYC	01-04	01-05	[C]	Validated	C4	Validated	V1	Compact	Booked	$20
S_3	1	NYC	01-04	01-05	[C]	CC Verified	C4	Validated	V1	Compact	Booked	
S_4	1	NYC	01-04	01-05	[C]	Unverified	C4	Validated	V1	Compact	Booked	-$20
S_5	1	NYC	01-04	01-05	[C]	Unverified	C4	Validated	V1	Compact	Canceled	
S_6	1	NYC	01-04	01-05	[C]	Unverified	C4	Validated	V1	Compact	Not Found	
S_7	1	NYC	01-04	01-05	[C]	Bonus	C4	Validated	V1	Compact	Not Found	-$10

Fig. 9. Secondary path: No vehicle is found and the customer receives a bonus

In this section, we described several secondary paths within the car rental scenario. Although our examples focus on a car rental scenario, the autogen-undo and autogen-redo procedures are widely applied in a variety of cancellation situations in several scenarios. Another common example is the cancellation of flights due to mechanical failures or unfavorable weather conditions. Due to the complex engineering of many recovery steps, long lines form at airports with passengers who would like to find the best alternative for their journeys. Similarly, long waits may happen with overloaded call centers when many recovery choices are possible. In order to improve the quality of recovery results, an optimization algorithm is added to autogen-redo and autogen-undo procedures, described in the following section.

4.3 Optimization Algorithm to find Max-profit Secondary Path

In this section, we describe the WED-flow support for incorporating an optimization algorithm into autogen-undo and autogen-redo procedures. As result, WED-flow recovery is able to find automatically the highest-profit secondary path (among all the alternatives that satisfy customer requirements), using the CR function as the objective function.

Consider an exception that occurred during the execution of the primary path, causing a violation of AWIC-consistency. WED-flow recovery (autogen-undo and autogen-redo, described in the previous section) can generate several secondary paths to recover AWIC-consistency. The main purpose of optimization algorithm is to find the secondary path that minimizes the cost (equivalently, maximizes the remaining profit). The optimization algorithm assumes that the component segments of secondary paths for recovery (both undo and redo) have been designed and implemented, ready for composition and execution by WED-flow. The search for appropriate component segments starts from the execution history (h) that contains the paths of active running instances. The search is refined by the data items ($assigned\_item$) affected by the exception, whose AWIC-consistency needs to be restored. Once all the relevant secondary paths (with $assigned\_item$) have been

1939-1374 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2824810, IEEE Transactions on Services Computing

found, the optimization algorithm chooses and executes the secondary path that has the lowest cost.

Fig. 10 outlines the optimization algorithm, in which the execution history (h) provides the context in which the secondary paths will be generated, and $assigned\_item$ identifies the parameters that affect the cost of recovery (in the car rental example it is the broken car). The first step of the algorithm finds the WED-states that form the context of optimization, determined by the scope of violation of AWIC-consistency, identified by the $assigned\_item$ in line 2. If no recovery instance is found, recovery actions are either unnecessary or unavailable, and the algorithm execution is terminated in line 4. Otherwise, the recovery algorithm enumerates the secondary paths found, and determine the most profitable secondary path. In lines 6-11, a loop iterates on instances in order to produce secondary paths by executing the autogen-undo and autogen-redo procedures. At each step of the iteration, the current state s_c and the stop point state s_{sp} are identified for each instance i. Both are obtained by queries on execution history (WED-states) of instance i in line 7 and 8. As discussed in Section 4.1, the stop-point state is used as stop condition for autogen-undo, and the current state is used as a stop condition for autogen-redo. For each secondary path created for each instance (lines 6-11), its cost/revenue is calculated using the CR function. After all secondary paths have been found, the highest profit path is identified and invoked in line 12.

```
1:  procedure RECOVERY(h, assigned_item)
2:      instances ← find_instances(h, assigned_item)
3:      if instances = ∅ then
4:          return false
5:      end if
6:      for i ∈ instances do
7:          s_c ← current_state(h_i)
8:          s_sp ← find_stop_point_state(h_i, assigned_item)
9:          autogen-undo(h_i, s_sp)
10:         autogen-redo(h_i, s_c)
11:     end for
12:     choose_highest_profit_path(instances)
13: end procedure
```

Fig. 10. Optimization algorithm to max-profit path

The autogen-undo procedure illustrated in Fig. 11 is responsible for recovering each instance from its current state to the stop-point state. The autogen-undo executes undo steps to semantically undo the effects produced by execution of WED-transitions between s_c and s_{sp}. The input parameters for autogen-undo are the execution history h_i of instance i, and the stop-point WED-state s_{sp}. In line 2, the autogen-undo procedure identifies the current state s_{nc} by executing a query on execution history h_i. Between lines 3-7, a loop executes until WED-state s_{nc} to be equivalent to stop-point WED-state s_{sp}. At each step of iteration, the WED-transition that produced WED-state s_{nc} is identified and its associated undo (sometimes a semantic compensation step) is executed, producing a new WED-state s_{nc}. The autogen-undo process terminates when a backward recovery path has been produced for instance i.

After autogen-undo execution, the autogen-redo procedure illustrated in Fig. 12 is invoked. Autogen-redo is responsible for finding a secondary path for each instance involved in the recovery process. This procedure has two input parameters: the execution history h_i of instance i, and WED-state s_c, which was the current state before starting autogen-undo. In line 2, the procedure gets the current state s_{nc} of h_i. The main purpose of autogen-redo is to construct the secondary paths that will produce a WED-state equivalent to s_c (returning to primary path), or a secondary path that produces an AWIC-consistent WED-state (final WED-state). Both stop conditions are used in loop in lines 3-6. At each step of iteration, autogen-redo finds a redo WED-transition to be executed by evaluating WED-state s_{nc} with WED-conditions of application. In line 5, the WED-transition associated to satisfied WED-condition is executed, and its execution produces a new WED-state s_{nc}. After autogen-redo execution, a secondary path will have been produced for instance i. Since many instances may be involved in the recovery process, we need to find which of them produces the max-profit secondary path as discussed below.

```
1:  procedure AUTOGEN-UNDO(h_i, s_sp)
2:      s_nc ← current_state(h_i)
3:      while s_nc ≢ s_sp do
4:          Let t be WED-transition that generated s_nc
5:          Let t^-1 be undo step for t;
6:          s_nc ← t^-1(s_nc)
7:      end while
8:  end procedure
```

Fig. 11. Autogen-undo procedure.

```
1:  procedure AUTOGEN-REDO(h_i, s_c)
2:      s_nc ← current_state(h_i)
3:      while s_nc ≢ s_c and not FINAL_STATE(s_nc) do
4:          Let t be WED-transition enabled by s_nc
5:          s_nc ← t(s_nc)
6:      end while
7:  end procedure
```

Fig. 12. Autogen-redo procedure

The recovery algorithm is used to produce a secondary path for each WED-flow instance involved in recovery. We use the concept of Nested Transactions [18] to avoid each secondary path has its effects exposed until the max-profit path has its data committed. Because of this, the algorithm described in Fig. 13 calculates the profit for each path using the CR function (see Definition 1). Next, in line 5, the instance k with highest profit secondary path is found and its data are committed (line 8). All other secondary paths are discarded (line 10). The recovery algorithm is encapsulated by a highest-level transaction while the execution of auto-undo and auto-redo (each secondary path) for each instance is enclosed in a sub-transaction. Finally, the lowest level refers to transitions and compensations that are performed encapsulated within transactions. An optimization to produce secondary paths is one of the ongoing works of our research group.

1939-1374 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2824810, IEEE Transactions on Services Computing


```
1: procedure CHOOSE_HIGHEST_PROFIT_PATH(instances)
2:     for i ∈ instances do
3:         profit_i ← CR(h_i)
4:     end for
5:     k ← max_profit(profit)
6:     for i ∈ instances do
7:         if i = k then
8:             commit(h_i)
9:         else
10:            abort(h_i)
11:        end if
12:    end for
13: end procedure
```

Fig. 13. Procedure for calculation and identification of the highest profit secondary path

The execution time for the optimization algorithm to the highest profit secondary path is characterized by two factors: (i) the number of instances involved in optimization process; and (ii) the number of compensations and transitions executed for each instance. In the next section, we present the experimental results obtained from different loads submitted to our algorithm.

5 Evaluation: A Feasibility Study of the Recovery

5.1 Implementation in the WED-flow framework

The algorithms described in Section 4.3 were implemented within the WED-flow framework. The WED-flow framework was developed using the Ruby programming language together with the Ruby on Rails application framework [11]. Both the process schema and the process instance are stored in a relational database. The framework features are grouped into modules as follows:

Definition and maintenance of process schemas. The process schema is designed in a XML file. This includes WED-attributes, WED-conditions, WED-transitions, WED-compensations and WED-triggers. Here, we include a new WED-attribute C/R used for WED-transitions and WED-compensations to store their costs and revenues. The XML file is used for the initial configuration of processes, which includes creation of the complete database structure.

Execution control. This module manages process instances and controls their execution. The module receives external events as initial attribute values (initial state) and begins a process instance to handle this new state. All the information about each new instance is recorded in a relational database. The execution control is responsible for execution of the primary path, in which the process control flow is determined by WED-conditions that are applied to the WED-state. Therefore, all WED-states must be monitored, and when one of them satisfies a condition, the associated transition will be triggered. More concretely, the concepts of WED-condition and WED-trigger were implemented through the fundamentals of continual queries [10].

Recovery management. This module is responsible for handling process instances when deviate from primary path. Such instances are interrupted by the execution control. The recovery management executes alternative actions such as backward and forward recovery steps. We included the optimization algorithm to max-profit secondary path within the recovery management module. This algorithm is triggered when an external event violates some AWIC-consistency rule.

5.2 Experimental setup

An example of business process was implemented based on the car rental business. The mechanism was submitted to 30 loads from 500 to 15,000 instances. We started the experiment with a load of 500 instances and added 500 instances for each subsequent load until the last load of 15,000 instances had been performed. For the sake of uniformity among the loads, each instance needed to undergo 3 to 5 steps in order to recover from the cancellation. Each load was submitted three times and the average execution times were recorded. We take care that each process instance has a viable secondary path (that is, it reaches a WED-state equivalent to the stop-point state or a final WED-state). If none of these alternatives was feasible, an inconsistent WED-state would be produced and the process instance would remain stuck. We conducted the experiments in a server running Ubuntu 14.04.02 LTS, PostgreSQL 9.3.10 and Ruby 2.3.0. The hardware used in the experiments included a server machine with 7 CPUs, Intel Xeon 2.4 GHz, 32 GB RAM and 200 GB disks.

5.3 Results

Fig. 14 depicts the experimental result obtained from execution of 30 loads. The first load with 500 instances was performed in 39.4 seconds, while the last load with 15,000 instances was performed in a runtime of 5,106.8 seconds.

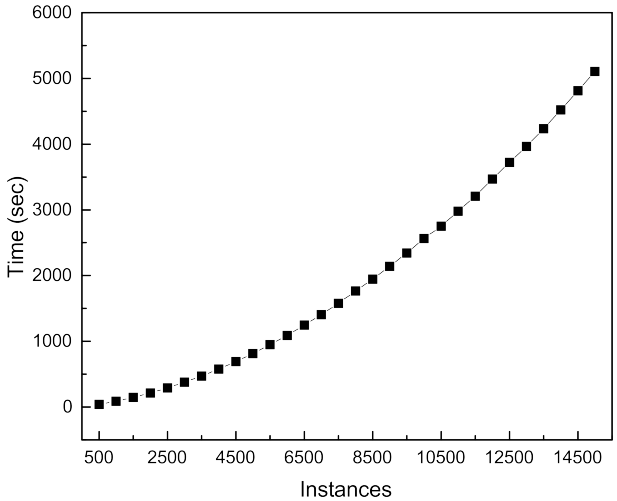


Fig. 14. Experimental results: number of instances versus time (sec).

The main objective of the feasibility study is to show that the cost-oriented recovery can be applied in different contexts even in scenarios involving a reasonable number of recovery transactional steps. The recovery complexity is evaluated by the number of backward and forward transactional recovery steps in which each instance involved in the recovery must perform to return to the consistency state.

1939-1374 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2824810, IEEE Transactions on Services Computing

Fig. 14 shows the behavior of the execution time given the addition of new recovery transactional steps. In each sample of our study, we increase the number of instances, and consequently the number of recovery transactional steps.

6 Related Work

The importance of robust and reliable PAIS has long been observed [19], [20]. Reliability is the ability of PAIS to provide efficient and accurate support for designing and implementation of automatic recovery routines for exception handling (i.e. expected exceptions). Robustness is the ability of PAIS to remain in operation despite unexpected exceptions. Robust and reliable PAIS are able to recover system from an inconsistent state to a consistent state when an abnormality causes deviation of execution.

The earliest work towards development of PAIS consisted of attempts to include advanced transactional concepts for supporting robust and reliable execution of business processes [21]. Among these, Garcia-Molina and Salem [8] were the first to introduce compensation actions associated with each step of the process for recovering consistency, in the Sagas model. The semi-atomicity proposal [22] extended the Sagas model by providing alternative execution paths that led the process to an acceptable final state after a failure. Opera [23] integrated programming language concepts and transactional atomicity, in which the semantics defined through the language constructs were enforced through use of an execution model based on extended transaction models (ETMs). Although many notable efforts have been made over the years to relax transaction properties in order to support reliable systems, contributions from ETMs only comprise a fraction of the many components needed for providing new abstractions towards robust and reliable processes [24], [25]. Bhiri et al. [25] argued that primitive control structures and non-intuitive sets of constraints imposed on structures were the main reasons for limited adoption of ETMs for supporting applications.

Transactional features were incorporated with traditional workflow to support business processes with well-defined failure semantics and recovery features were proposed in the early 1990s [20], [26], [27]. The ConTract Model [28], [29] ensured semantic database consistency by ensuring semantic rules known as invariants. Kamath and Ramamritham [30] proposed an approach called opportunistic compensation and re-execution that reduced recovery overheads when workflows were rolled back partially and re-executed to handle errors. Task Net [31] was a transactional workflow model based on a colored Petri net, which was used to express intra and inter-task state dependency. This prominent contribution provided clear evidence that oriented-process models still face challenges relating to reliability and flexibility that need to be overcome. The need for dependency mapping beforehand entails large overheads for analysts and designers and leads to complex process models.

As workflow technology consolidated, significant contributions regarding workflow recovery for ensuring reliable execution were also proposed. A classification framework for exception handling in PAIS based on patterns was proposed [32]. Exception handling patterns for process modeling were also addressed [33]. Adams et al. [34] proposed an extensible repertoire of self-contained exception handling processes (named exlets) based on workflow exception patterns and an associated set of selection rules to support dynamically exception handling for business process instances. A more comprehensive overview [32] evaluated several approaches with regard to their exception handing capabilities using a pattern-based approach. The Chimera-Exc language [35] was proposed for expressing exceptions in workflow management systems. Chimera-Exc is a language based on detached active rules that are used to capture exceptional events and to react to them.

Since Service-Oriented Architecture (SOA) has become a popular solution for distributed systems, several techniques have been proposed for robust and reliable Web service composition [36], [37], [38]. Although Web service compositions are beyond the scope of the present work, one of the most outstanding studies combines the coordination and organizational aspects of PAIS with the reliability of transactional processing [25]. A more detailed survey [39] has been published.

All of these approaches have significantly contributed towards improvement of process design and exception handling. However, to date, to the best of our knowledge, none of these new strategies has included an automatic recovery mechanism with the ability to find alternative paths for reducing the financial side-effects of exception handling.

Our solution to handle exceptions is supported by the WED-flow approach. This approach is an alternative to process modeling represented by the primitive constructors: triggers, conditions, transitions, and data state. An important phase for information systems modeling for processes is the capturing and ordering of events [40]. Usually, the events are ordered through temporal dependencies [41], and the data repository is modeled using classical database [42]. Event-driven methodology is a well-known strategy to identify and represent processes [41], [43]. However, the modeling of complex tasks goes beyond just capturing and representing events. In contrast to event-oriented strategies, the WED-flow approach is based on integrated paradigms (work, event, and data-flow) and concept of advanced transaction models. A more comprehensive comparison among the WED-flow approach and different process modeling paradigms (event-driven modeling, data-driven modeling, and control-flow modeling) may also be found in [6].

The main problem of PAIS in dealing with exceptions is their limited ability for modeling and automatic execution all possible outcomes and alternatives in a process. Because of this need, exception handling has usually been addressed through intervention of human operators in call centers, to solve the side-effects of unexpected events. Gradually, these ad-hoc interventions have become more common because of the exponential growth of systems. Therefore, maintenance of PAIS is becoming increasingly expensive and error-prone. To address this problem, our approach focuses on automating recovery routines that can be widely used in several business processes with the aim of reducing human operator intervention for exception handling. The feasibility study is intended to show that the recovery mechanism is appropri-

1939-1374 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2824810, IEEE Transactions on Services Computing

ate for practical purposes. The result illustrated in Fig. 14 indicates that the optimization algorithm to max-profit secondary path is feasible for execution within real business processes. Even though the response time for a large set of instances is substantial, our proposal is able to reduce significantly overheads relating to call centers, by providing an automated way for exception handling. In some business scenarios, not all instances may be involved in exception handling, because of semantic factors. For example, high-priority customers may not have their reservations canceled if there are other low-priority instances that can be canceled, even if this means a higher cost to the company.

Many PAIS are based on process models that provide well-founded notation that enables formal verification of correctness properties. These techniques avoid many serious structural problems in process models, such as deadlock and livelock. However, these are not enough to ensure robust and reliable execution when a particular process instance needs to deviate from the normal behavior defined by the flow structure. Because of this limitation, many PAIS are integrated with procedural languages in order to code specific routines for exception handling. In spite of the flexibility provided by these languages, exception handling code tends to grow much bigger than the primary path because of the many possibilities for errors. This significant code growth leads to various problems such as maintenance, testing and reuse. In contrast to these approaches, our proposal includes a reduced number of recovery steps that is enough for accurate and efficient management of exception handling. These recovery steps are homogeneously designed in accordance with the WED-flow framework. This facilitates their reuse, minimizes the effort involved in automation and reduces the complexity of exception handling for dynamic environments.

One requirement of PAIS in dynamic environments is that it should support long-duration execution of process instances that can become intertwined when sharing data resources. This generates dependencies among instances that have to be taken into consideration when dealing with an exception. An exception in this scenario can result in a side-effect that goes beyond the boundaries of the process instance in which the exception occurred. In addition, because the exception involves other instances, handling it within this scenario can also lead to significant costs for the business. As discussed in Section 4, our cost-aware recovery composition method is able to find secondary paths for all process instances that were indirectly affected by the exception. The instance that minimizes financial loss arising from exception handling can then be automatically selected. Intuitively, when an exception (violation of AWIC-consistency) is detected, our proposal allows automatic rearrangement of systems in order to comply with the dependencies among process instances. In contrast with another study [31], there is no need to map all dependencies at the design time, in our approach. We assume that the dependencies are generated dynamically when instances share data resources. Since the execution histories of process instances are stored permanently as data states, dependent instances affected by exception handling can be identified through a simple query on execution history, as shown in Section 4.3.

7 Conclusion

PAIS have serious difficulties with anomalous situations that fall outside of the well-behaved execution path. Hence, exception handling is usually delegated to call centers in order to recover application data consistency from failures. This results in significant additional costs for businesses. In this paper, we introduce a cost-aware recovery composition method that is able to find and follow secondary paths that reduce the cost of exception handling. From a practical point of view, our proposal reduces complexity and the need for manual interventions to handle exceptions.

Our WED-flow research initiative addresses the challenges of transactional recovery needed in complex scientific and business process. Our ongoing research includes development of an adaptation mechanism to deal with automatic migration of running process instances in adaptive PAIS. In addition, we have ongoing efforts towards ensuring correctness properties in WED-flow models that are able to avoid anomalies at the build time. Regarding to exception handling, we are studying some optimization alternatives by introducing parallel computing ideas into our recovery algorithm; a generic utility function to which many factors can be translated to replace the current cost function; and new strategies for comparative experiments and evaluation our work.

Acknowledgments

This work has been supported by FAPESP (São Paulo State Research Foundation) grant number 2015/01587-0, CNPq (Brazilian National Research Council) grant number 308476/2015-8, and Fundação Araucária.

References

[1] M. Dumas, W. van der Aalst, and A. H. M. ter Hofstede, Eds., *Process-Aware Information Systems: Bridging People and Software through Process Technology*. New York, NY, USA: Wiley, 2005.

[2] M. Weske, *Business Process Management: Concepts, Languages, Architectures*. Secaucus, NJ, USA: Springer-Verlag, 2007.

[3] J. Eder and W. Liebhart, "Contributions to exception handling in workflow management," in *Proc. EDBT Workshop Workflow Manage. Syst.*, 1998, pp. 3–10.

[4] Technavio, "Global contact center market 2015-2019," Technavio, London, UK, Tech. Rep., 2015.

[5] J. E. Ferreira, Q. Wu, S. Malkowski, and C. Pu, "Towards flexible event-handling in workflows through data states," in *Proc. 6th World Congr. on Services (SERVICES-1)*, 2010, pp. 344–351.

[6] J. E. Ferreira, O. K. Takai, S. Malkowski, and C. Pu, "Reducing exception handling complexity in business process modeling and implementation: the wed-flow approach," in *Proc. the Int'l Conf. the Move to Meaningful Internet Syst. (OTM'10)*, 2010, pp. 150–167.

[7] J. E. Ferreira, K. R. Braghetto, O. K. Takai, and C. Pu, "Transactional recovery support for robust exception handling in business process services," in *Proc. the 19th Int'l Conf. Web Services (ICWS 12)*, 2012, pp. 303–310.

[8] H. Garca-Molina and K. Salem, "Sagas," *ACM SIGMOD Rec.*, vol. 16, no. 3, pp. 249–259, Dec. 1987.

[9] D. B. Lomet, R. S. Barga, M. F. Mokbel, G. Shegalov, R. Wang, and Y. Zhu, "Immortal db: Transaction time support for sql server," in *Proc. the 2005 ACM SIGMOD Int'l Conf. Manage. of Data (SIGMOD '05)*, 2005, pp. 939–941.

[10] L. Liu, C. Pu, and W. Tang, "Continual queries for internet scale event-driven information delivery," *IEEE Trans. Knowl. Data Eng.*, vol. 11, no. 4, pp. 610–628, July 1999.

[11] M. Garcia, K. Braghetto, C. Pu, and J. E. Ferreira, "An implementation of a transaction model for business process systems," *J. Inform. and Data Manage. (JIDM)*, vol. 3, no. 3, pp. 271–286, 2012.

1939-1374 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2824810, IEEE Transactions on Services Computing

[12] J. E. Ferreira, O. K. Takai, and C. Pu, "Integration of business processes with autonomous information systems: a case study in government services," in *7th IEEE Int'l Conf. E-Commerce Technol. (CEC'05)*, 2005, pp. 471–474.
[13] L. V. Arajo, S. Malkowski, K. R. Braghetto, M. R. Passos-Bueno, M. Zatz, C. Pu, and J. E. Ferreira, "A rigorous approach to facilitate and guarantee the correctness of the genetic testing management in human genome information systems," *BMC Genomics*, vol. 12, no. 4, pp. 1–8, 2011.
[14] W. M. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. Barros, "Workflow patterns," *Distributed and Parallel Database*, vol. 14, no. 1, pp. 5–51, July 2003.
[15] N. Russell, A. H. M. T. Hofstede, W. M. van der Aalst, and N. Mulyar, "Workflow control flow patterns: A revised view," BPMcenter.org, Tech. Rep., 2006.
[16] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, "Greedy algorithms," in *Introduction to Algorithms*. Cambridge, MA, USA: MIT Press, 2009, pp. 414–450.
[17] S. Boyd and L. Vandenberghe, *Convex Optimization*. New York, NY, USA: Cambridge University Press, 2004.
[18] J. E. B. Moss, "Nested transactions: an approach to reliable distributed computing," Massachusetts Inst. of Technol., Cambridge, MA, USA, Tech. Rep., 1981.
[19] D. Georgakopoulos, M. Hornick, and A. Sheth, "An overview of workflow management: From process modeling to workflow automation infrastructure," *Distributed and Parallel Databases*, vol. 3, no. 2, pp. 119–153, Apr. 1995.
[20] J. Eder and W. Liebhart, "Workflow recovery," in *Proc. the 1st IFCIS Int'l Conf. Cooperative Inform. Syst. (CoopIS'96)*, 1996, pp. 124–134.
[21] A. K. Elmagarmid, Ed., *Database Transaction Models for Advanced Applications*. San Francisco, CA, USA: Morgan Kaufmann, 1992.
[22] A. Zhang, M. Nodine, B. Bhargava, and O. Bukhres, "Ensuring relaxed atomicity for flexible transactions in multidatabase systems," *SIGMOD Rec.*, vol. 23, no. 2, pp. 67–78, May 1994.
[23] C. Hagen and G. Alonso, "Exception handling in workflow management systems," *IEEE Trans. Softw. Eng.*, vol. 26, no. 10, pp. 943–958, Oct. 2000.
[24] G. Alonso, D. Agrawal, A. E. Abbadi, M. Kamath, R. Günthör, and C. Mohan, "Advanced transaction models in workflow contexts," in *Proc. the 12th Int'l Conf. Data Eng. (ICDE'96)*, 1996, pp. 574–581.
[25] S. Bhiri, C. Godart, and O. Perrin, "Transactional patterns for reliable web services compositions," in *Proc. the 6th Int'l Conf. Web Eng. (ICWE '06)*, 2006, pp. 137–144.
[26] A. Sheth and M. Rusinkiewicz, "On transactional workflows," *Data Eng. Bull.*, vol. 16, no. 2, pp. 20–25, 1993.
[27] D. Georgakopoulos, M. F. Hornick, and F. Manola, "Customizing transaction models and mechanisms in a programmable environment supporting reliable workflow automation," *IEEE Trans. Knowl. Data Eng.*, vol. 8, no. 4, pp. 630–649, Aug. 1996.
[28] H. Wächter and A. Reuter, "The contract model," in *Database Transaction Models for Advanced Appl.*, A. K. Elmagarmid, Ed. San Francisco, CA, USA: Morgan Kaufmann, 1992, pp. 219–263.
[29] A. Reuter, K. Schneider, and F. Schwenkreis, "Contracts revisited," in *Advanced Transaction Models and Architectures*, S. Jajodia and L. Kerschberg, Eds. Springer US, 1997, pp. 127–151.
[30] M. Karnath and K. Ramamritham, "Failure handling and coordinated execution of concurrent workflows," in *Proc. 14th Int'l Conf. Data Eng.*, 1998, pp. 334–341.
[31] I. Choi, C. Park, and C. Lee, "Task net: Transactional workflow model based on colored petri net," *Eur. J. Operational Res.*, vol. 136, no. 2, pp. 383–402, 2002.
[32] N. Russell, W. M. van der Aalst, and A. H. M. T. Hofstede, "Workflow exception patterns," in *Proc. the 18th Int'l Conf. Advanced Inform. Syst. Eng. (CAiSE'06)*, 2006, pp. 288–302.
[33] B. S. Lerner, S. Christov, L. J. Osterweil, R. Bendraou, U. Kannengiesser, and A. Wise, "Exception handling patterns for process modeling," *IEEE Trans. Softw. Eng.*, vol. 36, no. 2, pp. 162–183, 2010.
[34] M. Adams, A. H. ter Hofstede, W. M. van der Aalst, and D. Edmond, "Dynamic, extensible and context-aware exception handling for workflows," in *Proc. the 2007 OTM Confederated Int'l Conf. the Move to Meaningful Internet Syst. (OTM'07)*, 2007, pp. 95–112.
[35] F. Casati, S. Ceri, S. Paraboschi, and G. Pozzi, "Specification and implementation of exceptions in workflow management systems," *ACM Trans. Database Syst. (TODS)*, vol. 24, no. 3, pp. 405–451, 1999.
[36] C. ai Sun, E. el Khoury, and M. Aiello, "Transaction management in service-oriented systems: Requirements and a proposal," *IEEE Trans. Serv. Comput.*, vol. 4, no. 2, pp. 167–180, Apr. 2011.
[37] O. Bushehrian, S. Zare, and N. K. Rad, "A workflow-based failure recovery in web services composition," *J. Software Eng. and Appl.*, vol. 5, no. 2, pp. 89–95, 2012.
[38] R. Angarita, M. Rukoz, and Y. Cardinale, "Modeling dynamic recovery strategy for composite web services execution," *World Wide Web*, vol. 19, no. 1, pp. 89–109, 2015.
[39] Y. Cardinale, J. E. Haddad, M. Manouvrier, and M. Rukoz, "Transactional-aware web service composition: A survey," in *IGI Global-Advances in Knowledge Manage. (AKM)*, 2011, pp. 116–141.
[40] S. M. McMenamin and J. F. Palmer, *Essential Systems Analysis*. Upper Saddle River, NJ, USA: Yourdon Press, 1984.
[41] D. C. Luckham, *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Boston, MA, USA: Addison-Wesley, 2001.
[42] R. Elmasri and S. B. Navathe, *Fundamentals of Database Systems*, 5th ed. Boston, MA, USA: Addison-Wesley, 2006.
[43] N. Alexopoulou, M. Nikolaidou, D. Anagnostopoulos, and D. Martakos, "An event-driven modeling approach for dynamic human-intensive business processes," in *Proc. the Int'l Conf. Bus. Process Manage. - Workshops*, Ulm, Germany, 2009.

André Luis Schwerz received the PhD degree from Institute of Mathematics and Statistics at University of São Paulo, Brazil, in 2016. Currently, he is a professor at the Federal University of Technology - Paraná (UTFPR), Brazil, working in the Department of Computing. His research interests are in services computing for data science and engeneering.

Rafael Liberato received the PhD degree from Institute of Mathematics and Statistics at University of São Paulo, Brazil, in 2016. Currently, he is a professor at the Federal University of Technology - Paraná (UTFPR), Brazil, working in the Department of Computing.

Calton Pu received the PhD degree from University of Washington, Seattle, WA, in 1986 and served on the faculty of Columbia University and Oregon Graduate Institute. Currently, he is holding the position of Professor and John P. Imlay, Jr. Chair in Software in the College of Computing, Georgia Institute of Technology, Atlanta, GA. He has worked on several projects in systems and database research. He has published more than 70 journal papers and book chapters, 200 conference and refereed workshop papers. He served on more than 120 program committees. His recent research has focused on big data in Internet of Things, automated N-tier application deployment and denial of information. He is a fellow member of the IEEE.

João Eduardo Ferreira received the PhD degree from University of São Paulo, São Carlos, Brazil. In 2008-2009 he did his posdoc program in College of Computing at Georgia Tech, Atlanta USA. He has been researching in very large database modeling and integration since 1996. His main research interest is focused on transactions on services computing for data science and engeneering. He has published more than 40 journal papers and book chapters, 80 conference and refereed workshop papers. He served on more than 50 program committees. He is a faculty member in the Department of Computer Science, Institute of Mathematics and Statistics, at University of São Paulo, Brazil since 1999.

1939-1374 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.