An Implementation of a Transaction Model for Business Process Systems

Marcela O. Garcia[1], Kelly R. Braghetto[1], Calton Pu[2], João E. Ferreira[1]

[1] Institute of Mathematics and Statistics, University of São Paulo
{mortega, kellyrb, jef}@ime.usp.br
[2] Center for Experimental Research in Computer Systems, Georgia Institute of Technology
calton@cc.gatech.edu

Abstract. Business process systems have received increased attention from research and industry communities specially in order to guarantee transactional properties. Despite the significant efforts in new transactional models and software tools developed, the business processes still offer a scenario with several theoretical and practical challenges. In this article, we present an implementation of a transaction model based on WED(Work, Event and Data)-flow approach. WED-flow combines the concepts of business step composition, advanced transaction models, events, and data states in order to provide transactional support for business process systems. More concretely, our WED-flow implementation supports important aspects of business process systems such as: correct execution, integrity of business process instances in parallel execution, traceability, and recovery mechanisms.

Categories and Subject Descriptors: H.2 [**Database Management**]: Transaction Processing

Keywords: business process management, continual queries, long transactions, complex events

1. INTRODUCTION

Nowadays, the global outsourcing phenomenon has increased the demand to expose the internal business processes of enterprises as services and make them available in the Internet. Consequently, these modern business processes are done in heterogeneous environment that presents significant challenges to express service compositions with complex structure and ensure execution reliability. Many solutions in this context prioritize the control-flow and neglect data-flow of business processes. As consequence, they are not able to cover all requirements of service compositions because the preconditions for some business steps cannot be specified only in terms of pre-defined flows of tasks. Additionally, reliability problems can occur due to the absence of adequate transactional support in the business process context.

Business process modeling is the first step in BPM lifecycle and it is a complicated process that has been studied for many decades. Thus, a variety of approaches and products are available to support this phase, presenting different strengths and weaknesses. Lu and Sadiq stated that there are two most predominant approaches which have been used for modeling business processes: graph-based and rule-based [Lu and Sadiq 2007]. Using a graph-based approach, the process definition is composed by nodes, which represent the activities within a process, and arcs connecting these nodes, which represent control flow and data dependencies. In this way, the execution paths of the business process are explicitly declared. In a rule-based approach, the process logic is composed by a set of rules that specify activities properties such as the preconditions for their execution. Thus, the possible execution paths may be inferred according to the defined rules at runtime.

This work was supported by FAPESP (São Paulo State Research Foundation) grant numbers 2010/15493-4 and 2011/24114-0 and by CNPq (Brazilian National Research Council) grant number 201557/2009-6.

Currently, Workflow Management Systems (WFMSs) have been used to automate business processes, allowing their modeling, controlling their execution, and providing analysis. However, as indicated by [Alonso et al. 1996], WFMSs generally do not consider reliability issues and they are not able to ensure the correctness of the business process execution in case of failures. Aiming to address reliability problems, advanced transactional models have been proposed to provide flexible transactional properties for applications that consider conventional ACID properties rigid and unsuitable. The first important contribution that relaxes the limitations imposed by ACID properties was the saga model [Garcia-Molina and Salem 1987]. Saga is a consolidated transactional model that provides the concept of compensatory transaction. Following the direction of the saga model, but including more two types of transactional steps (pivot and retrievable), the notion of semiatomicity for flexible transactions was presented by [Zhang et al. 1994]. More recent contributions [Vidyasankar and Vossen 2011] and [Bhiri et al. 2011] proposed models to implementing business processes with transactional properties that are based on the concept of semiatomicity.

Despite the important contributions of all aforementioned approaches, the implementation of a business process model that captures complex structures and rely on transactional properties to guarantee reliability remains a great challenge. Placed in the context of rule-based models and transactional properties, this article presents the implementation of the WED-flow approach [Ferreira et al. 2012]. The WED-flow model is based on a set of pairs condition-transition which, at runtime, determine which activities will be executed in a business process instance and their execution order. These conditions are defined over data states of the process, and a satisfied condition enables the execution of its associated transition. Our implementation of WED-flow utilizes the concept of continual queries [Liu et al. 1999], which are able to monitor and capture changes in data states, and saga steps to create service compositions.

In the implementation exhibited in this article, we describe solutions to deal with challenges commonly found also in other transactional models for business process applications. These challenges can be summarized by the following items: 1) to guarantee correction in the execution of business processes; 2) to maintain the integrity of business process instances, where activities can be performed parallelly and the data states can be concurrently accessed and updated; 3) to relax the isolation properties of transactions, in order to enable the exposition of intermediate data states; 4) to guarantee the traceability of business processes (e.g., knowing when and under which circumstances business steps were triggered); 5) to support different recovery mechanisms, in order to deal with cancellations, system interruptions or exceptions during the execution of the activities of a business process; and 6) to provide a management system that can be easily integrated with the business applications.

Aiming overcome aforementioned challenges, our WED-flow implementation is based on relational database, which guarantees the persistence of business process elements and allows their reuse in different models. Furthermore, detailed data about each process instance and each transition execution are also stored in database relations, which provide support to recovery mechanisms. With regard to the business process execution, our implementation relies on advanced transactional models to provide transactional properties and ensure correct executions. In addition, the processes execution control is performed in a decentralized manner, allowing the use of our system in distributed environments.

The remainder of the article is organized as follows. Section 2 outlines the related work. The main WED-flow concepts are summarized in Section 3. Section 4 describes the implementation of the WED-flow approach. Section 5 shows a concrete example of a specific business process using WED-flow implementation. Section 6 concludes this article.

2. RELATED WORK

Business processes have been studied under varied modeling approaches, that can be grouped according different criteria. In the discussions made in this section, we will consider only two criteria, that are more closely related to the implementation approach we present in this article: (i) the expressive power for describing the process control flow, and (ii) the transactional properties provided by the modeling approach. Section 2.1 discusses some representative approaches under the first criterion, while Section 2.2 discusses models considered important under the second criterion.

2.1 Graph-based process models × Rule-based process models

Lu and Sadiq conducted a comparative study on business process modeling languages aiming to investigate the strengths and limitations of different theoretical foundations [Lu and Sadiq 2007]. In their research, they identified two most predominant formalisms that have been used to develop business process modeling languages: graph-based and rule-based formalism.

Mainly due to their legibility, graph-based process models are largely used in practice. As well-known examples of graph-based modeling languages specifically developed for the business process domain, we can cite the *Business Process Model and Notation* (BPMN) [OMG 2011], the *Event-driven Process Chains* (EPC) [Scheer et al. 2005] and the Activity Diagrams of the *Unified Modeling Language* (UML) [OMG 2010]. There are several works devoted to provide a formal semantics for these modeling languages, aiming to enable the verification and validation of the modeled processes. As example, we can cite the work of Aalst et al., related to the use o Petri nets in workflow modeling [van der Aalst 1998], and works related to the use of process algebra, such as the work of Pullman et al., that maps BPMN process models into specifications in a process algebra called Pi-calculus [Puhlmann 2007], and the work of Braghetto et al., which is based on Algebra of Communicating Processes [Braghetto et al. 2009].

Graph-based approaches models use graphical elements to specify the process definition, such as nodes representing steps of the process, and arcs between nodes to describe control flow and data dependencies. Thus, the process is precise and explicitly defined by a visual and intuitive language, which can be understood even for those who have little or no technical background [Lu and Sadiq 2007]. While the variety of analysis techniques provided by the precise definition of the business process is considered an advantage, the rigidity characteristic of graph-based models incurs problems of lack of flexibility, dynamism and adaptability, which compromise the model evolution and the ability of reacting to exceptional situations [Lu and Sadiq 2007].

The paradox of controlling processes and avoiding incorrect executions and also provide flexibility was addressed by [van der Aalst et al. 2009]. They proposed a declarative approach to balance between support and flexibility, which is based on constraints that determine that "anything is possible as long as it is not explicitly forbidden". Thus, there is no need to define execution paths explicitly; only the rules that constrain the behavior of the process need to be specified. An implementation that relies on finite state automatons was exhibited. The authors stated that the implementation presents performance limitation, having problems to deal with large specifications due to the complexity of the model-checking techniques. Rule-based models for business processes was also addressed by [Bry et al. 2006]. The authors advocate that Event-Condition-Action (ECA) rules "offer a flexible, adaptive, and modular approach to realizing business processes".

Rule-based modeling languages use rules to represent structural, data and/or resource dependencies between task executions in business processes. They presented high expressibility, being capable to express same constructs as those specified by graphical operators [Lu and Sadiq 2007]. Unlike graph-based process models, using rule-based approach, it is not necessary to define explicitly all possible execution paths of the business process. According with rules definitions (e.g., preconditions for activity execution), it is possible to infer and allow many execution paths at runtime. This flexible characteristic also favors the incremental evolution of the model and exception handling. However, specifying and managing a large number of rules for complex business processes require reasonable knowledge on rule-based formalism [Lu and Sadiq 2007].

2.2 Transactional Process Models

In order to guarantee a reliable execution, a WFMS must be able to assure transactional properties for the business processes it controls. A reliable management system must be able to provide recovery support and keep data consistency for process instances in case of failures occurred during their execution. The automation of a business process generally is made based on a process model defined in modeling languages as the ones we mentioned in Section 2.1. However, most part of these languages do not include mechanisms to appropriately guarantee transactional properties.

Several transactional models have been proposed in order to relax conventional ACID properties, that, in turn, are considered too restrictive to accommodate all the requirements of business process modeling. One of the first approaches for dealing with transactional processes is the saga model [Garcia-Molina and Salem 1987]. Garcia-Molina and Salem addressed problems related to long lived transactions, that are transactions which "hold on to database resources for relatively long periods of time, significantly delaying the termination of shorter and more common transactions". The proposed model considers a process composed by steps which are independent transactions (in the conventional database sense) that are executed sequentially or in parallel. The atomicity is achieved by the concept of compensation; the saga model assume that there is a compensating action for each step in the process. In case of an exception, the compensating steps for all steps that have been completed are executed. Thus, there are two execution options: the process terminates successfully or it is compensated.

Trying to solve the saga model imposition, in which the process terminates successfully or it is compensated, the concept of semiatomicity was proposed for flexible transactions [Zhang et al. 1994]. The primary idea of semiatomicity is to extend the saga model providing alternative execution paths. In this model, there are three types of transactional properties that characterize steps within a process: compensatable, pivotal, and retriable. A step is compensatable if it has a compensating step that semantically undoes its effects. A step is retriable if it is guaranteed to succeed if its execution is retried a sufficient number of times. A step is pivotal if it is neither compensatable nor retriable. Some models were presented using the semiatomicity idea. Schuldt et al. proposed "a unified model for concurrency control and recovery for processes" [Schuldt et al. 2002], and Vidyasankar and Vossen proposed a model with hierarchical-flexible transactions [Vidyasankar and Vossen 2011]. The primary contribution of semiatomicity is the use of alternative paths, which may be used to conduct the process to an acceptable final state after a failure. However, the modeling complexity increases significantly. The designer needs to identify steps as compensatable, retriable and pivotal, combine them, and also compose alternative paths that lead to a well-formed process [Alonso 2005].

Recently, Bhiri et al. proposed a Transactional Composite Service (TCS) model that "integrates the expressivity power and business adequacy of Workflow Systems and the reliability of Advanced Transactional Models", which is closely related to our work due to its overall goal. Using the concept of semiatomicity, each service belonging to the composite is characterized by a transaction property (i.e., compensatable, retriable or pivot). The proposed model is based on services states (e.g., initial, aborted, active, completed) and transitions (e.g., abort, activate, cancel, fail, complete), which determine the possible states which a service execution can go through, and the possible transitions between these states. The business process model is defined by dependencies between services; these dependencies are expressed by conditions over the services states (e.g., the completed state of a service enables the activation transition of another service). The implementation presented by Bhiri et al. uses a graphic interface to create the compositions that is based on workflow patterns. Besides the rigidity imposed by a graphic-based model, the need of expressing a condition over the services states limits the expressivity power of the model. In addition, as mentioned before, the modeling complexity is increased by the concept of semiatomicity.

The available solutions for modeling and implementation of business processes have significant limitations. The reasons of these limitations can be summarized by the absence of adequate transactional support, the prioritization of the control-flow (that results in "rigid" process models), and/or the neglect of data-flow. Our WED-flow implementation described in the following sections shows that it is possible to overcome these limitations and offer an alternative transactional support to model and implement flexible and reliable business processes.

3. WED-FLOW

WED(*Work, Event, Data*)-*flow* is an alternative approach for the modeling and implementation of business processes. Recently proposed in [Ferreira et al. 2010], WED-flow combines the concepts of business step composition, transactions, events, and data states, having as its main goal to reduce the complexity of exception handling. In the WED-flow approach, the instantiation of a business process is triggered by an event. This instantiation initiates the captures of changes in data states (by the verification of conditions defined over the data), and the triggering of data state transitions when appropriate, in similar manner to ECA rules. More concretely, when a new data state is provided as initial state for a WED-flow, this state will be evaluated by a collection of conditions, and when a condition is true, the associated transition is triggered, which will produce a new data state. The same idea is repeated for each new state, determining the control-flow of the WED-flow instance.

Our WED-flow implementation privileges the descriptions of conditions needed for each step in service compositions instead of how the compositions must be done. During the modeling phase, conditions and transitions must be specified, and using these definitions to create pairs condition-transition, structures responsible for condition evaluation and transition triggering are defined. Thus, it is possible to reuse conditions and transitions definitions to compose and design distinct WED-flows. Moreover, maintenance and evolution of the business process model can be performed by the addition or removal of conditions and transitions definitions. Similarly, when any exception becomes common, its treatment may be designed by a composition of conditions and transitions, generating new rules for the WED-flow.

When a business process is instantiated, the WED-flow system is responsible for transforming an inconsistent application state into a consistent application state by the execution of a set of state transitions. Each transition can be seen as a saga step, and during the execution of the saga steps of a WED-flow, we record all important execution information at each step boundary. This record includes data changes such as the old and the new values of data items modified by each saga step, and application-wide integrity constraints that need to be maintained by the WED-flow.

The principal definitions of WED-flow approach were presented in a previous work [Ferreira et al. 2012] and are summarized below.

—**WED-attributes**: The *WED-attributes* of an application are defined by the tuple $\mathcal{A} = \langle a_1, a_2, \ldots, a_n \rangle$, where each a_i (with $1 \leq i \leq n$) is an attribute of interest for the considered application.

—**WED-state**: A *WED-state* is a tuple $\langle v_1, v_2, \ldots, v_n \rangle$, where v_i (with $1 \leq i \leq n$) is a value for the i-th attribute of $\mathcal{A}$.

—**WED-condition**: A *WED-condition* is a set of predicates defined over the WED-attributes of an application. Let s be a WED-state and c a WED-condition. We say that s *satisfies* c if its attributes' values make the predicates of c true.

—**WED-transition**: A *WED-transition* is a function that receives as input a WED-state and generates as output a new WED-state. Its specification includes a set of database attributes $\mathcal{U}_t = u_1, u_2, \ldots, u_m$, such that u_i (with $1 \leq q \leq m \leq n$) is an element of $\mathcal{A}$. This set $\mathcal{U}_t$ indicates the set of attributes that are updated by the transition.

—**WED-trigger**: A *WED-trigger* is a 2-tuple $g = \langle c, t \rangle$, where c is a WED-condition and t a WED-transition. When a WED-state satisfies the condition c, g will trigger transition t.

—**WED-flow**: A *WED-flow* is a 3-tuple $\langle G, c_i, c_f \rangle$, where G is a set of WED-triggers, and c_i and c_f are WED-conditions. The condition c_i defines the set of valid initial WED-state(s), while c_f defines the set of valid final WED-state(s) for the flow. When a new WED-state is provided as initial state, a new *instance* of a WED-flow is created to handle this state.

—**AWIC-Consistent**: *Application-Wide Integrity Constraints* (AWICs) are integrity constraints that span all the autonomous databases used by an application. An AWIC can be defined as predicates involving the WED-attributes of the application. For convenience, we can define the AWICs of an application as a subset of the WED-conditions set.

—**Transaction-Consistent WED-state**: A WED-state s, which belongs to a WED-flow instance i, is a *transaction-consistent* WED-state if it respects at least one of the following properties:
—If s enables the triggering of at least one transition in the instance i;
—There is at least one transition currently in execution in instance i.

—**Inconsistent WED-state**: A WED-state is *inconsistent* if it is not AWIC-consistent, neither transaction-consistent.

—**History of a WED-flow Instance:** A history entry records the execution of a WED-transition and can be represented by the tuple $\langle s_c, t, s_i, s_o \rangle$, where:
—s_c indicates the WED-state that satisfied the condition that triggered t;
—t is the WED-transition executed;
—s_i indicates the WED-state provided as input to t;
—s_o indicates the WED-state generated as output of t.
A *history of a WED-flow instance* is a sequence $\langle e_1, e_2, \ldots, e_k \rangle$, where each e_i (with $1 \leq i \leq k$) is a history entry.

—**Current state**: We denote by *current state* of an instance the state s_o of the last entry in its history.

Considering the fundamentals presented above, we have developed an implementation of transaction model for business process systems, which is described in next section.

4. THE WED-FLOW SYSTEM CORE

In this section we describe a concrete implementation of the concepts presented in Section 3, which composes the WED-flow system core. The main purpose of the WED-flow core is to control the execution of transactional processes and provide the necessary structure to support recovery mechanisms and incremental evolution of exception handling. Our implementation has been developed using Ruby[1].

As described in a previous work [Ferreira et al. 2010], the design of a business process using the WED-flow approach consists of three main phases. First, the business process is separated into the critical path and the exception handling, and second, both parts are modeled by a composition of the four fundamental concepts (events, data, conditions, and transitions). Once the WED-flow model has been produced, the third phase consists in translating the model into a concrete specification language. In our implementation, the WED-flow model is described in a XML file; i.e., WED-attributes, WED-conditions, WED-transitions, and WED-triggers of the WED-flow model must be described using XML syntax. Then, this XML file will be used for the initial configuration of our system, which includes the creation of the complete database structure.

Concretely, our implementation is based on relational databases, and we are using PostgreSQL, an open source object-relational database system. The conceptual model of our database is represented in the entity-relationship (ER) diagram shown in Figure 1. This diagram models all the entities, relationships and constraints that can be extracted from the definitions presented in Section 3. The

[1]More details about our implementation are available on `http://data.ime.usp.br/wedflow`

logical model derived from this ER diagram is described in figures 2 and 3. In the database structure there are two groups of relations: those associated with the business process and those created to support the process execution. In the first group there are standard relations, whose structure does not depend on the business process model, that store information about WED-flows, WED-conditions, WED-transitions, and WED-triggers extracted from the XML file. There is also a database relation, directly related to the concepts of WED-attributes and WED-state, whose structure is defined in function of the business process being represented. The database relations associated with the business process are shown in Figure 2, and they are detailed below.

Each element of a WED-flow model is associated with a database relation. The relation *R_ WED-States*, whose attributes are specified in the WED-attributes set, stores all data states of each execution step. Thus, each WED-state is a specific instance (i.e. a row) in this relation. It is important to note that the definition of the WED-attributes and their management in our implementation generate an important side effect: a data replication between underlying database applications and our system. There are many approaches to treat this data replication problem in the database literature and it is out of the scope of this article to present the best solution for this problem. However, in the WED-flow approach we assume that the number of attributes that belong to underlying database applications and at the same time are under the WED-flow system core control is much smaller than the attributes that belong to all underlying database applications. Otherwise, the control of data replication would be impractical as well as WED-flow approach would not be plausible.

As WED-states, WED-conditions, WED-transitions, and WED-triggers are also associated with a database relation. To begin, each WED-condition is recorded in the relation *R_ WED-Conditions*, where the predicates are stored as an ordered list in the attribute *predicates*, and the complete condition expression is stored in postfix notation in *expression*. Additionally, the relation has an attribute which indicates if that WED-condition belongs to the AWICs set. The concept of WED-transition is mapped in two parts: an entry in a database relation with attributes that identify the transition, and a Ruby class that implements a well-defined interface, which must contains the execution code of the transition. Lastly, since a WED-trigger is a 2-tuple $g = \langle c, t \rangle$, this concept is mapped in a database relation with the id of WED-Condition c and the id of WED-transition t. Furthermore, all relations have an *id* attribute, and the attribute *active* indicates if that instance is active in our system.

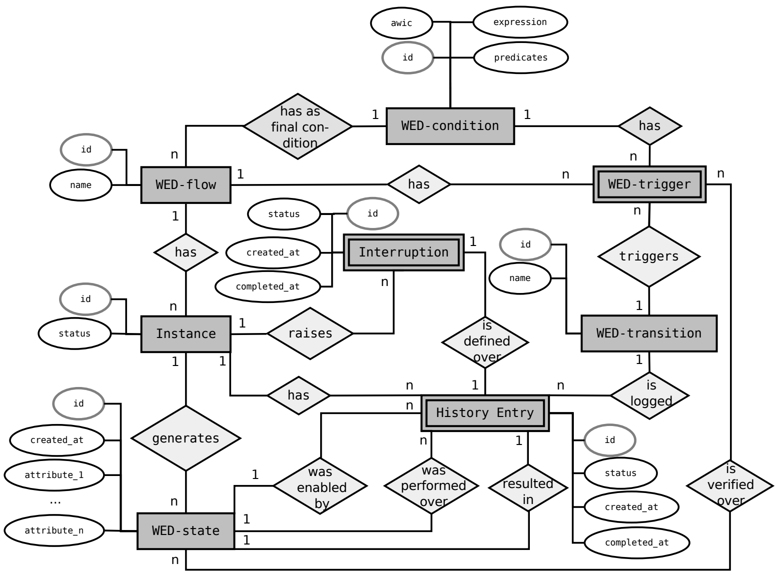

Fig. 1. Entity-relationship diagram.

R_WED-States

id	wed_flow_instance_id	created_at	a_1	...	a_n

R_WED-Conditions

id	name	predicates	expression	awic	active

R_WED-Transitions

id	name	active

R_WED-Flows

id	name	final_condition_id	active

R_WED-Triggers

id	wed_flow_id	wed_condition_id	wed_transition_id	active

Fig. 2. Database relations related to the business process.

The WED-flows information is stored in the database relation *R_ WED-Flows*. In our implementation, the initial condition (denoted by c_i in Section 3) is implicitly defined by the WED-state that is provided as initial state during the WED-flow instantiation. Therefore, it is not necessary to define the initial condition of a WED-flow. It is important to notice that we can create several different WED-flows for a same application. While the WED-triggers definitions are specific and directly associated with a WED-flow, WED-conditions and WED-transitions definitions are general and can be used in all WED-flows. Moreover, a WED-flow model can evolve incrementally through the addition or removal of WED-conditions, WED-transitions, and WED-triggers. For this purpose, our system provides an interface that allows the designer to use a XML file to create new, or disable current definitions, updating the model.

4.1 Execution control

After reading the specification of the WED-flow model and creating the necessary structure, the system is capable of receiving WED-flow instances to control their execution. In the scope of this work, a user must provide the initial values for an initial WED-state and also choose which WED-flow will be instantiated to handle the new state. Information about each new instance is recorded in relation shown in Figure 3 (a). In the WED-flow approach the control-flow is determined by WED-conditions that are applied to WED-state attribute values; therefore, all WED-states must be monitored, and when one of which satisfies a condition, the associated transition will be fired. More concretely, the concepts of WED-condition and WED-trigger were implemented through the fundamentals of continual queries.

A continual query has three main parts: a query, a trigger condition, and a termination condition [Liu et al. 1999]. In the WED-flow context, a WED-condition can be seen as a query, a trigger condition is expressed as a time frequency, and a termination condition is not required, since the WED-conditions must be checked while the WED-flow system is running. Moreover, instead of returning the query result to a user, the WED-flow system triggers the appropriate transitions. In the WED-flow system, each WED-trigger is responsible for checking its own condition and trigger the associated transition when necessary. In other words, when a WED-state s is generated in the application, each WED-trigger $g_j = \langle c_j, t_j \rangle$ must evaluate whether the WED-condition c_j is satisfied by the s values, and trigger the WED-transition t_j when the evaluation returns true.

Since WED-triggers must process each WED-state only once, we have implemented a queue of WED-states for each WED-trigger. In addition, each WED-trigger has an associated regular time value that determine the frequency at which the queue will be processed. This time value of each WED-trigger must be defined in the WED-flow model, and with those values the processing of each queue can be scheduled. For integrity purposes, all the queues are maintained in a database relation containing the WED-trigger id and the WED-state id. Thus, a new WED-state is inserted in those queues as soon as it has been generated, and when the clock event occurs, the information is retrieved

(a) R_WED-Flow_Instances

id	wed_flow_id	status

(b) R_Triggers_Queues

id	wed_trigger_id	wed_state_id

(c) R_Execution_History

id	wed_transition_id	wed_flow_instance_id	status	created_at	completed_at	initial_state_id	current_state_id	final_state_id

(d) R_Interruptions

id	wed_instance_id	interrupted_execution_id	status	created_at	recovered_at

Fig. 3. Database relations created to support the process execution.

ordered by the WED-states creation dates. The structure of the database relation used to control the WED-triggers queues is shown in Figure 3 (b) and it belongs to the aforementioned group of relations created to support the process execution. This relation is resultant from the mapping of the many-to-many relationship between the entities WED-trigger and WED-state, shown in the ER diagram of Figure 1.

After the instantiation of a WED-flow, its execution is started by offering the initial WED-state to all WED-triggers associated with the selected WED-flow (i.e., inserted in all queues), activating the WED-conditions evaluations that will enable the triggering of WED-transitions. Each WED-transition execution produces an output WED-state, the new current state of the instance, which may trigger the execution of other(s) WED-transition(s). It is important to note that there is just one current state in the process instance. When a new WED-state is produced by a WED-transition t, there is a specific set of database attributes that are updated by t, denoted by $\mathcal{U}_t$ in Section 3. This update is performed using the instance's current state as input. After producing the new WED-state, there are two options of offering this state: insert it in all queues, the most simple and obvious way, or select WED-triggers according to the updated attributes. In other words, using the second way, the new WED-state must be inserted in the queue of a WED-trigger only whether the associated WED-condition evaluates at least one attribute of $\mathcal{U}_t$. Although the first option is simpler, it produces longer queues which may cause performance issues. Moreover, the first approach can also cause undesired implications in modeling phase of parallel situations. To explain how the problem occurs, we will present a simple example that explores a scenario with transitions that can be triggered parallelly.

WED-attributes:

$\mathcal{A} = \langle a_1, a_2 \rangle$

WED-conditions:

$c_1 : (a_1 = X)$

$c_2 : (a_2 = Y)$

WED-transitions:

t_1 : updates a_1 with value X' $(a_1 \leftarrow X')$

t_2 : updates a_2 with value Y' $(a_2 \leftarrow Y')$

WED-triggers:

$g_1 = \langle c_1, t_1 \rangle$

$g_2 = \langle c_2, t_2 \rangle$

According to the presented model, the WED-state $s_0 = \langle X, Y \rangle$ triggers both transitions t_1 and t_2. If the transition t_1 is finished before t_2, its execution will produce a new state $s_1 = \langle X', Y \rangle$, which will trigger t_2 again, because s_1 satisfies c_1. In this way, the value of the attribute a_2 is unduly processed, i.e., the occurrence of the value Y in a_2 is processed twice instead of once. To avoid the problem, it would be necessary to combine both WED-conditions c_1 and c_2 in one with the configuration $c = (a_1 = X \text{ AND } a_2 = Y)$, and then, associate the new condition with transitions t_1 and t_2, creating two different WED-triggers. Considering that a parallel situation can involve several different conditions, their combination would increase modeling complexity, and consequently, the risk of modeling errors.

We believe that parallel scenarios are present in several business processes, therefore, our system must be able to control parallel execution paths. To address the problem described above, the second option of offering a new WED-state to WED-triggers was chosen. Thus, there is no additional complexity in modeling phase and the system performance is also improved. Utilizing the second approach, returning to the context of the presented example scenario, the initial WED-state s_0 will be offered to all WED-triggers and the other ones, produced later by transitions, will be offered according to the updated attributes. For example, $s_1 = \langle X', Y \rangle$ was produced by t_1, updating attribute a_1, and thus s_1 must be offered only to WED-trigger g_1, because its associated WED-condition c_1 evaluates a_1.

4.2 Recovery Support

The WED-flow approach combines the concepts of advanced transaction models, events, and data states to provide transactional recovery and incremental evolution of exception handling. The WED-flow core system must provide all structure and details to support the implementation of the recovery module, an ongoing activity.

To build a system capable to perform backward and forward recovery mechanisms, it is necessary to record the detailed data states of each execution step and also details about the associated transition execution. As mentioned before, all WED-states are carefully recorded in a database relation, and the execution details are directly related to the concept of history of a WED-flow instance. Besides the history entry (i.e., the 4-tuple $\langle s_c, t, s_i, s_o \rangle$), we also record timestamps and status information in the relation shown in Figure 3 (c). As soon as a WED-transition is triggered, a history entry is created with a 'fired' status, and when its execution is finished, the entry status is updated to 'completed'.

The goal of a WED-flow is to transform an inconsistent application state into a consistent application state by the execution of a set of state transitions, triggered by the WED-triggers [Ferreira et al. 2012]. During this process, an inconsistent state may be produced, and the recovery module must assume the responsibility to conduct the WED-flow instance to a transaction-consistent state. An inconsistent WED-state may be related to an interruption of a WED-transition execution or to a mistake in the WED-flow design, e.g., no WED-trigger was designed to handle the state, and consequently, the instance execution remains stuck.

A WED-transition interruption may occur due to a time-out, a manual cancellation, or to a writing conflict during the update of the current state. The last issue may occur in parallel executions, since that a WED-state that enables the triggering of a transition may not be the same state provided as input for this transition [Ferreira et al. 2012]. A WED-transition can update an attribute only whether its value has not been changed since the triggering of the transition, i.e., the attribute's value is the same in the WED-state that enabled the transition triggering and in the current WED-state of the instance. When the attribute's value has been changed by a parallel transition that has finished earlier, there is a writing conflict and the transition must be aborted.

In both cases of inconsistent WED-state occurrences, the WED-flow instance execution must be interrupted. Thus, the WED-flow core updates the instance status to 'interrupted' (stored in the *R_ WED-Flow_ Instances* relation) and also creates an interruption entry in the relation *R_ Interruptions*, shown in Figure 3 (d). Each entry contains details about the interruption of the WED-flow instance. When this interruption is related to a WED-transition execution that was interrupted, the entry records the id of this execution (stored in the relation *R_ Execution_ History*) in the attribute *execution_ interrupted*. Moreover, the entry in the *R_ Execution_ History* relation has its status changed to 'aborted'. With the information stored in *R_ Execution_ History* and *R_ Interruptions* relations, the recovery module is capable to restore the application consistency.

5. A BUSINESS PROCESS EXAMPLE IN THE WED-FLOW SYSTEM

To illustrate how the WED-flow approach works, a simple example was presented in [Ferreira et al. 2012]. The WED-flow designed refers to the processing of travel requests in a travel agency, and we will use the same model to exemplify our implementation. In this agency, when a travel request is received, the first step is validate it. After that, the hotel can be reserved and the air ticket can be purchased, parallelly. If both activities are completed successfully, the mentioned travel request is closed. Appendix A shows the XML file related to the model, which will be used as an input file for our system.

The information about the WED-flow and its elements, WED-Conditions, WED-Transitions, and WED-Triggers extracted from the XML file is stored in database relations, as shown in Figure 4. Considering the declared WED-attributes, the relation that will store WED-states is created, as shown in Figure 5, which also contains data of the initial WED-states for our example.

To exemplify the business process execution, we will use two WED-flow instances, identified by their id number 1 and 2. The initial state of the first instance is denoted by p_0, and s_0 refers to the second instance. Both of them are described below and Figure 5 shows how data of new instances are stored in the database relations. This figure has three parts: part (a) shows the initial WED-states, part (b) shows the initial instances data, and, since the initial states must be inserted in the queues of the WED-triggers associated with the WED-flow, part (c) shows the database relation that represents these queues.

Instance 1:

$p_0 = \langle$ 1111, 'Not validated', NULL, 'Not requested', NULL, 'Not requested', 4444, 'Received' $\rangle$

Instance 2:

$s_0 = \langle$ 5555, 'Not validated', NULL, 'Not requested', NULL, 'Not requested', 8888, 'Received' $\rangle$

According to the specification of our example, shown in Appendix A, the processing frequency of the trigger with id = 1 is five seconds. Therefore, its queue, which contains WED-states 1 and 2, will be the first to be processed. The WED-condition *c_new_travel_requested* is evaluated on each state of the queue, and since this condition is satisfied by both WED-states 1 and 2, the WED-transition *t_validate_travel_request* will be triggered for both WED-flow instances. For instance 1, the WED-state 1 is the state that satisfied the condition and also the state used as input to *t_validate_travel_request*, producing the WED-state whose id is 3. For instance 2, WED-state 2

R_WED-Conditions

id	name	predicates	expression	awic	active
1	"c_new_travel_requested"	- customer_status,=,not validated - order_status,=,received	1 2 and	FALSE	TRUE
2	"c_hotel_requested"	- hotel_status,=,requested - order_status,=,validated	1 2 and	FALSE	TRUE
3	"c_air_ticket_requested"	- air_ticket_status,=,requested - order_status,=,validated	1 2 and	FALSE	TRUE
4	"c_request_treated"	- hotel_status,=,reserved - air_ticket_status,=,purchased	1 2 and	FALSE	TRUE
5	c_order_finalized	- customer_status,=,validated - hotel_status,=,reserved - air_ticket_status,=,purchased - order_status,=,finalized	1 2 and 3 and 4 and	TRUE	TRUE

R_WED-Transitions

id	name	active
1	"t_validate_travel_request"	TRUE
2	"t_reserve_hotel"	TRUE
3	"t_buy_air_ticket"	TRUE
4	"t_close_travel_request"	TRUE

R_WED-Triggers

id	wed_flow_id	wed_condition_id	wed_transition_id	active
1	1	1	1	TRUE
2	1	2	2	TRUE
3	1	3	3	TRUE
4	1	4	4	TRUE

R_WED-Flows

id	name	final_condition_id	acitve
1	travel_agency	5	TRUE

Fig. 4. Database relations with data extracted from the XML file.

(a) R_WED-States

id	wed_flow_instance_id	created_at	customer_id	customer_status	air_ticket_id	air_ticket_status	hotel_id	hotel_status	order_id	order_status
1	1	"12-05-24 15:51:00"	1111	"not validated"	NULL	"not requested"	NULL	"not requested"	4444	"received"
2	2	"12-05-24 15:51:00"	5555	"not validated"	NULL	"not requested"	NULL	"not requested"	8888	"received"

(b) R_WED-flow_Instances

id	wed_flow_id	status
1	1	"active"
2	1	"active"

(c) R_Triggers_Queues

id	wed_trigger_id	wed_state_id
1	1	1
2	2	1
3	3	1
4	4	1
5	1	2
6	2	2
7	3	2
8	4	2

Fig. 5. New Instances.

satisfied the condition and is also used as input for the transition execution that produces the WED-state whose id is 4. The database relation *R_ WED-States* at this moment is shown in Figure 6 (a). Details about both executions are shown in Figure 6 (b).

The WED-transition *t_ validate_ travel_ request* updates the attributes *customer_ status, order_ status, air_ ticket_ status, hotel_ status*. Thus, the new state produced by this transition, like WED-states 3 and 4, must be inserted in all WED-triggers queues, since the WED-condition associated with each WED-trigger in our example evaluates at least one of those attributes. Figure 7 shows the database relation that represents the WED-triggers queues, and it is important to notice that the entries with id 1 and 5 were removed, since the queue of the WED-trigger with id = 1 was processed. Furthermore, eight new entries were created due to the insertion of WED-states 3 and 4.

After ten seconds of system startup, the second processing of WED-trigger 1 will occur, along processing of WED-triggers 2 and 3. The queue of WED-trigger 1 contains WED-states 3 and 4, but, since those states do not satisfy WED-condition *c_ new_ travel_ requested*, no transition will be triggered. Both queues of WED-triggers 2 and 3 contain WED-states 1, 2, 3 and 4 and the latter two will be responsible for triggering transitions as detailed below.

During processing of WED-trigger 2, WED-states 3 (from instance 1) and 4 (from instance 2) will trigger WED-transition *t_ reserve_ hotel*, because both states satisfy WED-condition *c_ hotel_ requested*. Processing of WED-trigger 3 will trigger *t_ buy_ air_ ticket* for instances 1 and 2 from WED-states 3 and 4, respectively. WED-transitions *t_ reserve_ hotel* and *t_ buy_ air_ ticket* are executed parallelly, and for instance 1, the execution of *t_ reserve_ hotel* ends earlier, producing WED-state 5 with a hotel id = 3333 and hotel status changed to "reserved". Thus, the transition that buy the air ticket will use WED-state 5 (the current state of instance 1) as input to produce the new WED-state with

(a) R_WED-States

id	wed_flow_instance_id	created_at	customer_id	customer_status	air_ticket_id	air_ticket_status	hotel_id	hotel_status	order_id	order_status
1	1	"12-05-24 15:51:00"	1111	"not validated"	NULL	"not requested"	NULL	"not requested"	4444	"received"
2	2	"12-05-24 15:51:00"	5555	"not validated"	NULL	"not requested"	NULL	"not requested"	8888	"received"
3	1	"12-05-24 15:51:05"	1111	"validated"	NULL	"requested"	NULL	"requested"	4444	"validated"
4	2	"12-05-24 15:51:06"	5555	"validated"	NULL	"requested"	NULL	"requested"	8888	"validated"

(b) R_Execution_History

id	wed_transition_id	wed_flow_instance_id	status	created_at	completed_at	initial_state_id	current_state_id	final_state_id
1	1	1	"success"	"12-05-24 15:51:05"	"12-05-24 15:51:06"	1	1	3
2	1	2	"success"	"12-05-24 15:51:05"	"12-05-24 15:51:06"	2	2	4

Fig. 6. Effects of WED-transition *t_ validate_ travel_ request* executions.

R_Triggers_Queues

id	wed_trigger_id	wed_state_id
2	2	1
3	3	1
4	4	1
6	2	2
7	3	2
8	4	2
9	1	3
10	2	3
11	3	3
12	4	3
13	1	4
14	2	4
15	3	4
16	4	4

Fig. 7. WED-triggers queues after the first processing of WED-trigger 1.

id 6. All WED-states can be seen in Figure 8 (a), which shows relation *R_ WED-States* after the end of instances execution. Figure 8 (b) shows relation *R_ Execution_ History* with details about the executions of WED-transitions.

For instance 2, the state produced by WED-transition *t_ reserve_ hotel* has id 7, and during the execution of *t_ buy_ air_ ticket*, occurs an interruption, and consequently, no new state is generated. Due to the interruption, an entry is created in relation *R_ Interruptions* as shown in Figure 9. Moreover, the status of the entry that represents the execution of transition *t_ buy_ air_ ticket* for instance 2 is updated to "aborted". This entry has id 5 in relation *R_ Execution_ History* and it can be seen in Figure 8 (b). As mentioned before, the interruption information is recorded to support transactional recovery. Thus, instance 2 will be stuck until recovery mechanisms handle it.

After more five seconds, processing of WED-trigger 4 will evaluate WED-condition *c_ request_ treated* on each WED-state in its queue: 1, 2, 3, 4, 5, 6 and 7. WED-state with id 7 is the only state that will trigger the WED-transition *t_ close_ travel_ request*, producing the new WED-state with id 8 for WED-flow instance 1. After that, the execution of instance 1 is finished, since WED-state 8 satisfies WED-condition *c_ order_ finalized*, which is the condition that identifies a final data state in the WED-flow "Travel Agency".

Figure 10 shows the sequence of WED-states for the execution of instances 1 and 2, identified by (a) and (b) respectively. Each WED-state is labeled with S-*i*, where *i* is the WED-state's id. Arrows represent states transitions. Solid black lines indicate completed executions of WED-transitions, and they are labeled with WED-transition's id, connecting the state that triggered the transition with the

(a) R_WED-States

id	wed_flow_instance_id	created_at	customer_id	customer_status	air_ticket_id	air_ticket_status	hotel_id	hotel_status	order_id	order_status
1	1	"12-05-24 15:51:00"	1111	"not validated"	NULL	"not requested"	NULL	"not requested"	4444	"received"
2	2	"12-05-24 15:51:00"	5555	"not validated"	NULL	"not requested"	NULL	"not requested"	8888	"received"
3	1	"12-05-24 15:51:05"	1111	"validated"	NULL	"requested"	NULL	"requested"	4444	"validated"
4	2	"12-05-24 15:51:06"	5555	"validated"	NULL	"requested"	NULL	"requested"	8888	"validated"
5	1	"12-05-24 15:51:10"	1111	"validated"	NULL	"requested"	3333	"reserved"	4444	"validated"
6	1	"12-05-24 15:51:10"	1111	"validated"	2222	"purchased"	3333	"reserved"	4444	"validated"
7	2	"12-05-24 15:51:11"	5555	"validated"	NULL	"requested"	7777	"reserved"	8888	"validated"
8	1	"12-05-24 15:51:16"	1111	"validated"	2222	"purchased"	3333	"reserved"	4444	"finalized"

(b) R_Execution_History

id	wed_transition_id	wed_flow_instance_id	status	created_at	completed_at	initial_state_id	current_state_id	final_state_id
1	1	1	"success"	"12-05-24 15:51:05"	"12-05-24 15:51:06"	1	1	3
2	1	2	"success"	"12-05-24 15:51:05"	"12-05-24 15:51:06"	2	2	4
3	2	1	"success"	"12-05-24 15:51:10"	"12-05-24 15:51:10"	3	3	5
4	3	1	"success"	"12-05-24 15:51:10"	"12-05-24 15:51:11"	3	5	6
5	3	2	"aborted"	"12-05-24 15:51:10"	"12-05-24 15:51:16"	4		
6	2	2	"success"	"12-05-24 15:51:10"	"12-05-24 15:51:11"	4	4	7
7	4	1	"success"	"12-05-24 15:51:16"	"12-05-24 15:51:16"	6	6	8

Fig. 8. Relations after the end of instances execution.

R_Interruptions

id	wed_flow_instance_id	interrupted_execution_id	status	created_at	recovered_at
1	2	5	“open”	“12-05-24 15:51:16”	

Fig. 9. Interruptions relation.

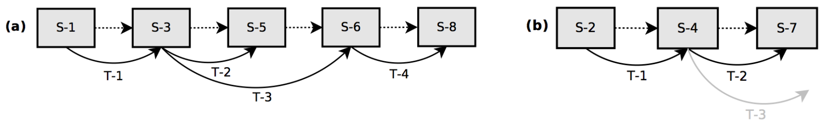

Fig. 10. WED-states sequences for the execution of two WED-flow instances.

output state. Dotted lines indicate the order that states are produced in these instances. The gray line shows the WED-transition *t_ buy_ air_ ticket* that was interrupted, and consequently, the WED-state S-7 is the last state of instance 2, since this instance is stuck waiting for recovery mechanisms.

6. CONCLUSION

Despite the considerable number of theoretical transaction models for business process applications we can found in literature, the management systems developed to support business processes defined using these models are scarce in practice. The reason behind this fact is the difficulty to implement such tools guaranteeing transactional properties for the executed processes. More concretely, this difficulty is due to different aspects that must be aimed in the implementation, such as: correct execution, integrity of business process instances for parallel execution, traceability, and recovery mechanisms.

In this article we presented a concrete implementation of WED-flow approach that supports the modeling and implementation of business processes in order to provide transactional properties for business process systems. The main purpose of the WED-flow system core is to control the execution of transactional processes and provide the necessary structure to support recovery mechanisms and incremental evolution of exception handling. This WED-flow system core covers the most important implementation aspects in order to provide an appropriate relaxation of isolation properties for business process applications. A famous business process (i.e., travel agency) was also presented in this article in order to illustrate how the WED-flow implementation solves some important and complex events of data and associates them with transactional properties.

In the overall picture of WED-flow implementation, the supporting for transactional requirements imposed by business process applications is the main contribution of this article. In addition, we have several ongoing efforts to improve the WED-flow system. First, we are including recovery mechanisms to treat the interruption of WED-flow instances. Second, we are developing a model checker to avoid the generation of inconsistent WED-states in the WED-flow design. Third, we are creating a methodological approach to support business specialists in the identification of relevant data events and in the design of the business processes.

APPENDIX A. XML EXAMPLE

The following XML file contains the specification of the business process presented as example in Section 5. This file is used as input to the WED-flow system, and we are using a XML Schema to describe the structure of the document, which is available on `http://data.ime.usp.br/wedflow` .

```
<?xml version="1.0" encoding="UTF-8"?>
<WED-flow-initial-schema>
  <WED-attributes>
    <Attribute Name="customer_id" Type="integer" />
    <Attribute Name="customer_status" Type="string" />
    <Attribute Name="air_ticket_id" Type="integer" />
    <Attribute Name="air_ticket_status" Type="string" />
    <Attribute Name="hotel_id" Type="integer" />
    <Attribute Name="hotel_status" Type="string" />
    <Attribute Name="order_id" Type="integer" />
    <Attribute Name="order_status" Type="string" />
  </WED-attributes>
  <WED-conditions>
    <Condition Name="c_new_travel_request" >
      <Predicate Id="1"> customer_status = ‘Not Validated’ </Predicate>
      <Predicate Id="2"> order_status = ‘Received’ </Predicate>
      <Expression> 1 AND 2 </Expression>
    </Condition>
    <Condition Name="c_hotel_requested" >
      <Predicate Id="1"> hotel_status = ‘Requested’ </Predicate>
      <Predicate Id="2"> order_status = ‘Validated’ </Predicate>
      <Expression> 1 AND 2 </Expression>
    </Condition>
    <Condition Name="c_air_ticket_requested" >
      <Predicate Id="1"> air_ticket_status = ‘Requested’ </Predicate>
      <Predicate Id="2"> order_status = ‘Validated’ </Predicate>
      <Expression> 1 AND 2 </Expression>
    </Condition>
    <Condition Name="c_request_treated" >
      <Predicate Id="1"> hotel_status = ‘Reserved’ </Predicate>
      <Predicate Id="2"> air_ticket_status = ‘Purchased’ </Predicate>
      <Expression> 1 AND 2 </Expression>
    </Condition>
    <Condition Name="c_order_finalized" >
      <Predicate Id="1"> customer_status = ‘Validated’ </Predicate>
      <Predicate Id="2"> air_ticket_status = ‘Purchased’ </Predicate>
      <Predicate Id="3"> hotel_status = ‘Reserved’ </Predicate>
      <Predicate Id="4"> order_status = ‘Finalized’ </Predicate>
      <Expression> 1 AND 2 AND 3 AND 4 </Expression>
    </Condition>
  </WED-conditions>
  <WED-transitions>
    <Transition Name="t_validate_travel_request" >
      <UpdatedAttribute AttrName="customer_status" />
      <UpdatedAttribute AttrName="order_status" />
      <UpdatedAttribute AttrName="air_ticket_status" />
      <UpdatedAttribute AttrName="hotel_status" />
    </Transition>
    <Transition Name="t_reserve_hotel" >
      <UpdatedAttribute AttrName="hotel_status" />
      <UpdatedAttribute AttrName="hotel_id" />
    </Transition>
    <Transition Name="t_buy_air_ticket" >
      <UpdatedAttribute AttrName="air_ticket_status" />
      <UpdatedAttribute AttrName="air_ticket_id" />
    </Transition>
    <Transition Name="t_close_travel_request" >
      <UpdatedAttribute AttrName="order_status" />
    </Transition>
  </WED-transitions>
  <WED-flows>
    <Flow Name="travel_agency" FinalStateCondName="c_order_finalized">
      <Trigger CondName="c_new_travel_request"
               TransName="t_validate_travel_request" Period="5" />
      <Trigger CondName="c_hotel_requested"
               TransName="t_reserve_hotel" Period="10" />
      <Trigger CondName="c_air_ticket_requested"
```

```
                TransName="t_buy_air_ticket" Period="10" />
      <Trigger CondName="c_request_treated"
                TransName="t_close_travel_request"  Period="15" />
    </Flow>
  </WED-flows>
  <AWICs><Constraint CondName="c_order_finalized" /></AWICs>
</WED-flow-initial-schema>
```

REFERENCES

Alonso, G. Transactional business processes. In *Process-Aware Information Systems*, M. Dumas, W. M. P. van der Aalst, and A. H. M. ter Hofstede (Eds.). John Wiley & Sons, Inc., pp. 257–278, 2005.

Alonso, G., Agrawal, D., Abbadi, A. E., Kamath, M., Günthör, R., and Mohan, C. Advanced transaction models in workflow contexts. In *Proceedings of the Twelfth International Conference on Data Engineering.* New Orleans, USA, pp. 574–581, 1996.

Bhiri, S., Gaaloul, W., Godart, C., Perrin, O., Zaremba, M., and Derguech, W. Ensuring customised transactional reliability of composite services. *J. Database Manag.* 22 (2): 64–92, 2011.

Braghetto, K. R., Ferreira, J. E., and Pu, C. NPTool: Towards scalability and reliability of business process management. In *e-Business and Telecommunications*, J. Filipe and M. S. Obaidat (Eds.). Communications in Computer and Information Science, vol. 48. Springer Berlin Heidelberg, pp. 99–112, 2009.

Bry, F., Eckert, M., lavinia Pătrânjan, P., and Romanenko, I. Realizing business processes with ECA rules: Benefits, challenges, limits. In *Proc. Int. Workshop on Principles and Practice of Semantic Web.* Budva, Montenegro, pp. 48–62, 2006.

Ferreira, J. E., Braghetto, K. R., Takai, O. K., Malkowski, S., and Pu, C. Transactional recovery support for robust exception handling in business process services. In *Proc. of the 19th Int. Conference on Web Services.* Honolulu, USA, pp. 303–310, 2012.

Ferreira, J. E., Takai, O. K., Malkowski, S., and Pu, C. Reducing exception handling complexity in business process modeling and implementation: the WED-flow approach. In *Proc. of the 18th Int. Conference on Cooperative Information Systems.* Hersonissos, Crete, Greece, pp. 150–167, 2010.

Ferreira, J. E., Wu, Q., Malkowski, S., and Pu, C. Towards flexible event-handling in workflows through data states. In *Proceedings of the 2010 6th World Congress on Services.* Miami, Florida, USA, pp. 344–351, 2010.

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

Liu, L., Pu, C., and Tang, W. Continual queries for internet scale event-driven information delivery. *IEEE Trans. on Knowl. and Data Eng.* 11 (4): 610–628, July, 1999.

Lu, R. and Sadiq, S. A survey of comparative business process modeling approaches. In *Proc. of the 10th international conference on Business information systems.* Poznan, Poland, pp. 82–94, 2007.

OMG. Unified modeling language specification (UML), version 2.3, 2010.

OMG. Business process model and notation (BPMN), version 2.0, 2011.

Puhlmann, F. Soundness verification of business processes specified in the pi-calculus. In *Proceedings of the 2007 OTM Confederated international conference on On the move to meaningful internet systems: CoopIS, DOA, ODBASE, GADA, and IS - Volume Part I.* Vilamoura, Portugal, pp. 6–23, 2007.

Scheer, A.-W., Thomas, O., and Adam, O. Process modeling using event-driven process chains. In *Process-Aware Information Systems*, M. Dumas, W. M. P. van der Aalst, and A. H. M. ter Hofstede (Eds.). John Wiley & Sons, Inc., pp. 119–145, 2005.

Schuldt, H., Alonso, G., Beeri, C., and Schek, H.-J. Atomicity and isolation for transactional processes. *ACM Trans. Database Syst.* 27 (1): 63–116, 2002.

van der Aalst, W. M. P. The application of Petri nets to workflow management. *Journal of Circuits, Systems, and Computers* 8 (1): 21–66, 1998.

van der Aalst, W. M. P., Pesic, M., and Schonenberg, H. Declarative workflows: Balancing between flexibility and support. *Computer Science - R&D* 23 (2): 99–113, 2009.

Vidyasankar, K. and Vossen, G. Multi-level modeling of web service compositions with transactional properties. *J. Database Manag.* 22 (2): 1–31, 2011.

Zhang, A., Nodine, M., Bhargava, B., and Bukhres, O. Ensuring relaxed atomicity for flexible transactions in multidatabase systems. *SIGMOD Rec.* 23 (2): 67–78, 1994.