A Novel Solution to Workflow Exception Handling for Adaptive Workflows in SOA

Tran Khanh Dang, An Sinh Le Trong, Minh Than Quang
Faculty of Computer Science & Engineering
HCMC University of Technology, VNUHCM, Vietnam
khanh@cse.hcmut.edu.vn, {thanqminh, ansinhletrong}@gmail.com

Abstract. As businesses all over the world change frequently and are specialized, companies take the approach of using other companies’ services to build their own ones. Hence, the information systems supporting those companies are naturally implemented using the newly emerged Service-Oriented Architecture (SOA). However, when the SOA-based business processes come to play, it becomes harder and costs higher to handle business exceptions (BEs). In this paper, we introduce a new approach of using data alternatives and control alternatives to effectively handle many kinds of BEs at the database level. In addition, along with our approach, we introduce a new exception-oriented process to develop SOA-based workflows, which facilitates the handling of BEs.

Keywords. Workflow exception, SOA, dynamic workflow change, workflow evolution, adaptive workflow database, WfMS.

1 Introduction

Nowadays companies are in a world that is flatted by the advances of technologies, making their businesses changed much more frequently than ever. In that context, there are new demands that all companies have to meet to remain competitive [3]:

- **Agility**: the ability to rapidly adapt to the changes in the business requirements, as well as to new business opportunities.
- **Responsiveness**: the ability to be more interactive with customers, and trading partners.
- **End-to-end visibility**: the ability to monitor and optimize performance in real time.

These demands lead to the use of the newly emerged Service-Oriented Architecture (SOA) to build companies’ business systems. Such systems are composed of four layers [4]: user interface, business process, services, and internal systems on which the services are built. In such an architecture, due to many reasons (wrong input data, business evolution, network errors, system shut down suddenly, etc.) exceptions may occur in all the four layers, yielding heavy costs on many aspects to the company [1, 5]. Loosing customer contracts, process delays, exception-handling costs (around $50 billion dollars per year worldwide), are those to count as typical examples.

In the abovementioned architecture, the business process layer differentiates one company from another. The business processes in this layer are built on top of SOA services, which are called **SOA-based business processes**. As a company’s business
evolves, there are exceptions appearing on this layer, causing headache to most system designers. Such exceptions can be classified as multi-version exceptions and bypassing exceptions [17], adaptive-workflow exceptions [6,7,9,10,11,12], or dynamic change bugs [8]. To deal with those vital business exceptions, most companies are currently ill-prepared and therefore usually use some pragmatic approach as an after-thought when exceptions occur [1].

On the today’s market, there are products that facilitate the handling of business exceptions, but they either support just a specific type of business exception, or are difficult to extend. In this paper, we introduce a new approach that support handling a variety of exception types, and recommend a novel exception-oriented process used to develop business processes.

The rest of this paper is organized as follows. Section 2 assesses existing workflow exception-handling techniques in SOA. Section 3 discusses why exceptions may occur in business processes. In section 4, we present our main contribution, a novel solution to deal with those exceptions. The last two sections evaluate the applicability of our solution, give concluding remarks of the paper and present the future work.

2 Related Works

For computers to be able to help, business processes must be modeled as workflows. A workflow is composed of steps (named activities) that relate to each other by temporal or causal relationships. Each step is corresponding to a step in the business process. All these steps and relationships compose a specification of a business process, which is called a workflow-schema. When there are customer requests, each customer will be served by an instance of the workflow-schema, which is called a workflow instance. Workflow schemas and workflow instances are managed by special systems called Workflow Management Systems (WfMS). Currently, there are four main strategies [12] to handle business exceptions in WfMS. Each strategy has its own weakness, and when used may generate more problems:

i. Flush the system. That means waiting for the system to process all in progress workflow instances, then bringing the new business process into execution.

ii. Abort all on-going jobs. That means cancelling all the workflow instances that are running, and then replace them by the new business process.

iii. Let the old and new versions of the business process coexist. That means many versions of a business process may be in use simultaneously.

iv. Adapt old workflow to new one. That means once the new business process is defined; the old business process must be migrated correctly to the new one.

The strategy (i) can only be applied on very small systems, and therefore is not realistic nowadays. With workflows that are not long running, and do not take high costs to re-run its steps, the approach (ii) may work well, but the current businesses are not that simple. Therefore, the strategy (iii) and (iv) are much more widely used.

The strategy (iii) may suite on processes whose semantics are basically unchanged, hence in the business process’s transition time (the time when customers for old and new business process coexists), many versions of the business process may be used to
serve many different “versions” of customers. This approach generates the problem of multi-version exception on business process [17]. The strategy (iv) is also known as the “adaptive WF approach”. That is, the business process should be defined in such a way that facilitates the migration of the old business process to the new one. The approach causes a new problem of defining proper correctness criteria to be used along with the migration of business processes. When this approach is used, all old running business processes will be “transformed” to an instance of the new business process; therefore, no multi-version exceptions can be found. In addition, it is not the matter strategy (iii) or (iv) are chosen, there always has the cases in which specific steps of a business process are skipped under certain conditions. This is called bypassing exceptions.

In [17], a database schema is defined to be able to capture the schema, the states of a business process, and also contains the definitions of contexts under which multi-version and bypassing exceptions occur, as well as holding lately defined tasks to deal with those exceptions. Hence, the approach is called handling business exceptions at the database level. However, the proposed approach is not general enough to be able to handle problems accompanied with the strategy (iv).

In [6, 7, 9, 11], different correctness criteria are proposed to make sure that the migrations of the old business process to the new one are correct. Using one of those correctness criteria, bypassing exceptions are eliminated during the process of migrating old business process to the new one. In general, this approach uses an application built on top of an existing system to do the migration; thus, it may be classified as handling business exceptions at the application levels. However, accompanying with the migration are dynamic change bugs [8] that are difficult to find and eliminate.

Another approach to be mentioned is building a data object model to monitor and handle business exceptions [14]. The data object model is used as a database to hold definitions of business process. Each step in the process has a responsible agent, initial assumptions, final goals, initial tests, invariant tests, and final tests. All the information is used to check for the occurrence of an exception. When the exception appears, an exception object is created and passed up the calling stack to find the appropriate exception handlers. This approach sounds fine, but has the problem of enormous costs on exception monitoring. Moreover, if exception handlers are not well defined in advance, the system may still be corrupted by business exceptions.

Also handling business process exceptions at the application level, there is an approach that uses outer processes (processes outside the business process) to handle exceptions [15]. An outer processes specialized in handling exceptions is called an exlet. The set of exlets defined to handle different exceptions on a business process is called an exlet pool. Whenever an exception occurs, the exception context is passed to an appropriate exlet chosen from the exlet pool. The exlet processes the context and then return the valid data for the business process to be able to continue smoothly. Because an exlet and the business process are unrelated, the approach yields the need of exlet logs. Business process logs case identifier to complete the operation history of the business process. Moreover, to be able to choose the right exlet, the approach adopts the use of an exception service. This exception service must track relationships, data mappings, and synchronization between business cases. Hence, the approach is neither sufficient nor able to support a wide range of exception types. In the next section, we introduce the new concepts of data alternatives and control alterna-
tives, as well as the use of these concepts to provide a general solution to the problem of business process exceptions.

3 Workflow Exceptions

As mentioned in section 2, a workflow consists of a number of activities which process the information of the business process in the form of data objects. Workflow is a combination of control flows and data flows. A control flow of a workflow is an order in which its activities are executed. A data flow is the flow of data objects processed through a control flow. Figure 1 demonstrates those concepts.

![Fig. 1. Data flow and control flow in a workflow](image)

In a workflow, exceptions may occur in both control flow and data flow. An exception in the control flow, called control-flow exception, is a change in the number of steps, or on the steps themselves, or in the order in which steps are executed. Similarly, changes to data items or data objects against the designed ones are considered data-flow exceptions. Those exceptions happen at some particular workflow instances. Most of other related works handle exceptions at application level because they focus on control-flow exceptions and do not specify the data flow. Our approach analyzes both the control flow and data flow to design an exception-aware workflow database.

Exceptions in business processes may be classified into two types: unpredictable exceptions and exceptions that occur due to the evolution of business processes. For the first type of exceptions, a careful analysis on business processes can reduce unpredictable cases. Handling the second type of exceptions is much more difficult. In most of the cases, a change in business processes originates from a change in the data flows of a workflow, i.e. the change may occur at some data objects in some particular activities. The changes could be due to a new definition of data objects or data items, unnecessary information, or the replacing of a known data object by a new data object or a new data item. If new data appears, usually we need a new step or modify some steps in the control flow in order to be able handle it.

To clarify this situation, we take examples of a real-world workflow in this and next sections that supports students to register the subjects they want to study in a semester in a university’s Course-Registration System (CRS). Although the workflow has a number of activities, we just pay attention to the prone-to-exception part of this
workflow: the activities that check for the subject constraints on registered subjects. Subject constraints are the relations among subjects in a curriculum of a training programme, i.e., there are some subjects that depend on some others. For examples, if subject X is the foundation of subject Y, a student has to study the subject X first in order to be allowed to register for subject Y. Typically, if student A registers for subject Y, but he has not studied the subject X yet, the system should detect that the subject constraint between X and Y has been violated and it will not allow him to register Y. However, if A has a certificate Z proving that he has enough knowledge to understand Y, and the head of department providing the subject Y confirms A’s case, then the system should allow A to study the subject Y. This situation could be considered as a bypassing on the data flow of the workflow, as mentioned in [17]. In this paper, we call Z a data alternative for X (see Figure 2, Case 1).

In another case, K2005 and K2006 refer to students starting their study in 2005 and 2006, respectively. From the K2006, the curriculum introduces new concepts of subject group and subject group constraint. A subject group consists of a number of subjects. A student will pass a subject group if he passes at least a certain number of subjects in the subject group. A subject group constraint is similar to the subject constraint we mentioned. In this case, a newly defined data leads to the use of an additional step on the control flow of the workflow to check for subject group constraints, yielding the need of using activity alternatives (see Figure 2, Case 2).

**Fig. 2.** Two examples of alternatives

4 An Extensible Exception Handling Approach at Database Level

In the previous section, we showed that most workflow exceptions caused by the evolution of business processes. Furthermore, in a workflow, most of business process changes occur in the data flow and may cause changes in the control flow. Based on this idea, we introduce two new concepts to model these changes: data alternative and activity alternative.

A data alternative is a data object or data item that is used as an alternative to the existed data object or data item in some particular workflow instances. Data alternatives may be classified into four types: a data object alters a data object, a data item object alters a data object, a data object alters a data item, and a data item object alters a data item. A data alternative’s structure may be similar to the original one, but this
is not a constraint, i.e. it can be completely different. Similarly, an activity alternative is an activity or a part of workflow, called subflow, that could be used instead of an old activity or subflow of some workflow instances. Figure 3 shows a workflow with its possible data alternatives and activity alternatives.

In quality management, the earlier errors are detected, the cheaper handling costs will be. We adopt that philosophy in our approach. In order to handle the exceptions that may occur in business process efficiently, we should identify the parts which are prone to the evolution of the workflow. Here is the first step in our approach: when designing a complete workflow based on a business process, the workflow designer should predict the part that may change in the future. Because most exceptions originate from the changes of data, the workflow designer has to determine which data may be changed to a new one. Later, when there are changes in the business process, those changes will be incorporated into the business process under the guise of data alternatives and activity alternatives. By that way, the evolution will be made seamlessly.

This first step is particularly important that determines the exception-handling capability of the workflow. That is why we call our method exception-oriented process. In next step, as usual, we model the business process as a workflow. However, we have to mark the data objects, which may evolve in the future, as adjustable data objects. Similarly, those steps that may evolve in order to process new data should be modeled as adjustable activities. Then we put into the workflow a number of alternative checking points (ACP), which is mainly used to dealing with exceptions may occur later.

When the workflow instance executes to an ACP, it will check whether an adjustable data object has been altered yet. If the data object is the old one, the workflow will continue its tasks as usual. Otherwise, it will search the exception database for a data alternative to alter the corresponding adjustable data object. If a data alternative was found, the engine will check whether the data alternative leads to the use of an activity alternative. If there is no such activity, then the system will treat the new data alternative as the original one because no new activity needed to process the new data object. If an activity alternative was found, the system will alter the corresponding adjustable activity by the activity alternative to process the new data object correctly.

Obviously, an important requirement for an ACP is that it must be placed into the workflow right before the adjustable activity in order to be able to replace the old one with the activity alternative. After our workflow executes for a period of time, the associated business process may changes, leading to exceptions. As we have already

![Fig. 3. Data alternatives and activity alternatives](image_url)
analyzed the workflow carefully, almost other instances work as expected. The exceptions should occur only in some special instances of the workflow. To treat those special cases, the system must be provided additional information about the new data objects as well as the activities definition needed to process them.

Fig. 4. Adaptive Workflow Database

In order to be able to store the definitions of data alternatives and activities alternatives, as well as the contexts under which those alternatives are used, we have designed a database schema called Adaptive Workflow Database (see Figure 4). Once a business process adopts our approach, the Adaptive Workflow Database will drive the handling of exceptions.

The database holds the definitions of workflows in the entity named “Assemblies”. Assembly is the common name for the compiled files such as DLL (Dynamic Link
Library), EXE (execution), etc. Once we have the assemblies in hand, all needed information about a workflow’s schema can be easily derived using reflection. Our approach of storing assemblies instead of storing the meta-data of workflows (the solution chosen by many other researchers) is more compact, and uses less memory.

Moreover, in the database, there are entities that contain information about activities and data objects or object-data items that are under consideration. The entity Activities store details about an activity, which might be an adjustable activity in a certain workflow, or might be an activity alternative. The entity DataObjects and ObjectDataItems store information about the adjustable data of a workflow, as well as information about the data alternatives. Because of the fact that we have classified data alternatives into many types as stated at the beginning of this section, the contexts of using data alternatives are hold in those entities with the suffix “_Alternatives” in their names. Such a context contains the GUID of the adjustable data object (aliased as original data), the GUID of the data object that is to be used as data alternatives (aliased as alternative data), the value of the original data, and the value of the alternative data. The pair of values helps determine when the context surfaces during the lifetime of a workflow instance. All those contexts are abstracted as data alternatives (stored in the entity DataAlternatives).

Contexts of using activity alternatives are stored in the entity ActivityAlternatives. Such a context contains the name of the adjustable activity (aliased as original activity), the name of the activity that is used as an activity alternative (aliased as alternative activity), and the GUID of the data alternative that triggers the use of the activity alternative. With all those information in the database, using the process previously stated, we can handle many types of exceptions as well as the contexts under which those alternatives are used.

A look at examples will clarify our approach. Because the additional data may cause exceptions, the CRS should collect the registration form as well as other certificates and forms from students if they were submitted. When designing the workflow schema, those additional documents should be marked as adjustable data objects, because they will be processed in other ways than normal. If the curriculum changes, a student of the previous course might have to take a new subject, leading to an omission of some old subject(s). The curriculum code of a student and his subject result list should be indicated as an adjustable data object because when it turns out to be unusual, the CRS should check the subject constraints in a different way. Moreover, we should also mark the activity that checks for subject constraints as an adjustable activity.

For the first example (cf. Figure 2, Case 1), when student A submits a certificate that confirms the knowledge of a subject X, the system administrator should add the information into the exception database that A’s certificate is a data alternative object for the subject result X. In this case, there is no need for an activity alternative because the CRS should consider this data object the result of subject X, therefore the workflow should work as originally designed. In the next example (cf. Figure 2, Case 2), if a new curriculum contains subject groups and subject group constraints, the activity that check for subject constraints should be changed to the new one, reflecting the evolution of the workflow. In this case, the administrator should add both the data alternative information for the new course year and the activity alternative that checks for single and group subject constraints into the database. As the workflow arrives at the ACP, it may find out that the course year is a new one, then will find the
activity alternative to process the subject result list. As we have seen, after we spend some efforts at the early stage of workflow designing, the cost of exception handling reduces dramatically by using the information from the Adaptive Workflow Database.

5 Assessments and Discussions

When using our approach to deal with exceptions at the database level, the appearances of workflow instances that contain exceptions are similar to the original one, because the system administrator does not modify a workflow directly. In fact, our workflow can be considered as a database-driven one, i.e. the execution of the workflow depends on the information stored in the Adaptive Workflow Database.

As we have seen in the two examples, after we spend some efforts at the early stage of modeling business processes, the cost of handling exceptions later reduces dramatically thank to the use of the Adaptive Workflow Database. We can see that a good preparation can result in an effective handling of exceptions. Spending a small effort for checking for exception at ACPs, our approach can deal with exceptions of many types in both data-flow and control-flow. Hence, it is a more general method than other previously introduced approaches.

When exceptions appear more often reducing the efficiency of the workflow, the system administrator should consider an upgrade on the system. Also, in the case of adaptive workflow (as mentioned in Section 2), if all steps that compose a workflow are marked as adjustable activities, the problem is easily solved by our approach.

6 Conclusion and Future Work

Among other approaches, in this paper, we have focused on handling exceptions at the database level. Furthermore, to support exceptions effectively, we require the workflow designer to be exception-oriented when analyzing the business process. As driven by the Adaptive Workflow Database, the effort to handle exception later is much reduced and the workflow is flexible enough to deal with exceptions. Although our approach is obviously a general method for handling exceptions and can be applied in many current WfMS, we have only currently implemented the approach using Microsoft Windows Workflow Foundation and it works smoothly.

In the future, we have planned to implement our approach on other WfMSs in order to establish the practical value of the newly created solution. In addition, we also want to develop a guideline for workflow designers who adopt this newly proposed solution to effectively create exception-aware workflow. Besides, applying the created solution to the batch processing of a vast number of business processes optimally [2] will also be of our great interest in the future.
References