Towards an Extensible Exceptions-Aware Database
Schema for Adaptive Workflows in SOA

Tran Khanh Dang, Viet Hung Nguyen, Hai Ly-Hoang

Advances in Security & Information Systems Lab
Faculty of CSE, HCM University of Technology
National University of Ho Chi Minh City, Vietnam
{khanh, hungnv, lhhai}@cse.hcmut.edu.vn

Abstract. The development of web technologies and service-oriented architecture (SOA) has leveraged workflow-aware system. Although a number of previous work have been carried out, current technologies are immature to support workflow exceptions in SOA-based system effectively. Whereas most of recent work are at the application level, our efforts are to introduce a new approach that will deal with such exceptions at the database level and can be applied in a variety of real-world application domains. In this paper, we propose an extensible exceptions-aware database schema for adaptive workflows in SOA. This new schema aims at supporting multi-version and bypassing problems in SOA-based workflow-processing systems. Our new approach is also a generalization of previously introduced ones.

Keywords: Adaptive workflow, exception-aware database schema, multi-version workflow, workflow bypassing, SOA.

1 Introduction

Nowadays, process engineering and process improvement are vital to the success of organizations. Standardized and managed processes can help to cut unprofitable expenses, reduce the mistakes, improve the effectiveness and assure stability in the quality of products as well. Therefore, many quality standards like ISO, CMM assess organizations based on the maturity of the processes established within those organizations. In this context, process automation is becoming important to organizations. Actually, workflow technology has been widely applied in recent years to automate well-defined and repetitive business processes. It has been deployed and well accepted in a great diversity of environments such as software engineering, healthcare, finance, production, office automation and banking [11].

In process automation, a business process is represented by a workflow. It comprises of a series of activities constrained by several temporal or causal relationships. The specification of a workflow is called workflow schema. It may be

---

1 This work is completed with the financial support of BKIS project (B2006-20-07TD), funded by National University of HCMC, and Ministry of Science & Technology, Vietnam.
instantiated in order to represent a performing occurrence of a business process. The result of such an instantiation is named workflow instance. A workflow management system (WfMS) is a generic information system that supports modeling, execution, management and monitoring of workflows [6].

One of the most concerned problems of workflow management system is the flexibility. Business processes are not rigid, but they keep evolving. Process improvement, the enactment of new policy/law, the change of the organization structure, etc. may lead organization to the modification of business processes. Besides, in case of exceptional cases, the original workflow has to be customized as well. Therefore, the WfMS must be flexible enough to empower the user to handle those changes. Recently, there have been many efforts to enhance the flexibility of workflow management system. Most of them focus on proposing a powerful, flexible workflow specification language or the propagation of changes in workflow schema to in-progress workflow instances. However, in our point of view, it is important that WfMS possesses adequate information so that it can fully support exception handling and dynamic changes of workflows. In this paper, we propose a novel comprehensive database schema, which maintains information about workflows. This schema is extensible to cover typical situations of dynamic changes within workflows. When there are new situations, based on the information kept in the schema, the system can be expanded to adapt to new requirements without impacts on the on-going workflows.

The rest of this paper is organized as follows. Section 2 briefly summarizes the state-of-the-art of adaptive workflow research. Section 3 details typical workflow exceptions. Next, section 4 presents our main contributions, an extensible exceptions-aware database schema for adaptive workflows in SOA-based information systems. Finally, section 5 gives concluding remarks and presents our future work.

2 Related Work

During the lifetime of a workflow instance, it is possible that exceptions may happen. An exception can be a system failure or an event that deviates the workflow from its original schema. A flexible WfMS has to support the user to handle those situations. In [1,2,3,5], workflow designers are responsible for defining a set of ECA rules (Event-Condition-Action) which help WfMS detect and handle known exceptions automatically. However, whether exceptions are handled manually or automatically, it is important that a WfMS must be designed as highly flexible as possible so that the workflow can be changed dynamically to adapt to exceptions and new situations. This is a major research topic in the research community. So far, existing solutions can be classified into two categories: Open Points or Dynamic Evolution [13]. Open Points approach defines points in workflow schema where adaption can be made. This approach is based on “late-modeling” to postpone the modeling task until the runtime of a workflow instance. The schema, which is used to create the workflow instance, is just a generic schema. During runtime, based on the current situation, the user has to choose/define a concrete implementation for generic steps of the original generic schema. In Dynamic Evolution approach, workflow instances are created from a
concrete schema at the beginning. However, during the execution, if workflow exceptions appear, they can be changed to adapt to the new situation. The adaption can be the assignment of the workflow instance directly to the new schema (e.g., [4]) or through a series of primitive change operations (e.g., ADEPT [13],[10]).

In order to dynamically adapt the workflow, the WfMS must be provided with enough information about the workflow schema and execution state of the workflow instance. One essential problem is to figure out a database schema which can handle all changes. Traditionally, a relational database schema is used for storing information about workflows. The structure of workflow schema (a control flow and a data flow), the execution state of workflow instance, and information about activities within workflows are stored as tables in relational database.

TRAM [7] and WASA 2 [9] maintain information supporting the execution of workflow and adaption process based on object-oriented approach. Workflow schemas, workflow instances are objects. The control and data flows are also represented in objects. In [9], the author sketches a class diagram presenting the association relationship between those objects. In that diagram, a workflow schema object and a workflow instance object are bound together by an instance-of association relationship which can be changed dynamically during the runtime of workflow instance. Hence, during its lifetime, a workflow instance can be controlled by various workflow schemas.

Windows Workflow Foundation (WWF) [12] is one of four pillars of Microsoft .NET 3.0/3.5 platform. It is the technology for defining, executing, and managing workflows [14]. In WWF, rather than persisted as a set of activities with associated control and data flow constraints, workflows are expressed in form of a piece of codes. A workflow designer is in charge of writing a method which builds up the workflow instance at runtime. Those codes can be written in any .NET-targeted languages. This way provides a great flexibility in defining workflow. However, the current WWF has not provided the ability of dynamically changing the workflow instance.

Traditionally, a workflow is represented in form of flowchart or its extensions. Alternatively, it can be considered as a state machine. In this way, a workflow is represented as a series of states linked by transitions. It starts from an initial state and ends at a final state. Transitions between states are represented by rules (in [8], rules are stated in JECA (Justification-Event-Condition-Action) form). The workflow schema of [8] consists of a set of rules and directed graphs representing the dependencies between them.

3 Workflow Exceptions

For the sake of clarity, we present examples in this and next sections in the context of a university information system. This system manages most of important workflows, especially ones spanning across many departments, in a university. These workflows are often presented as flowcharts built from the scratch and staffs’ experiences. Therefore, they are not well-defined, and able to change regularly during their lifetime. Even though these processes are stable, they could encounter some
exceptions in operation for any reason, such as special cases or changes of law. For instance, in the graduating approval process (GAP), there are many seniors who have delayed one or more semesters. If prerequisite course list, a graduating condition, has been changed, the delayed seniors who have not yet finished these courses will not be able to graduate due to missing new mandatory subjects.

Workflow exceptions mainly fall into following categories: control flow and data flow. Exceptions of a control flow change the number of steps, some extra steps need to be performed and/or some others are omitted; and/or they could change the flow’s direction. Those of data flow change the number of input/output data and/or replace data with other data. These changes happen at some particular instances of a workflow. Most of research work are only concerned with the changes of the control flow. They usually handle exceptions theoretically and do not provide detail about what kind of data stored within processes. Hence exceptions are managed at the application level.

Such systems have to deal with ‘unstable’ workflows, whose control flows are changed regularly as well as data flows, at the database level. Exceptions are analyzed carefully to incorporate the database design. This is the main philosophy of our approach. For unsolved problems, application-level mechanisms can be employed and dynamically attached to running systems later as other approaches do [7],[9],[10],[13].

In this paper, we deal with two kinds of exceptions: multi-version and bypassing problems. They could happen in both control and data flows. The multi-version problem of the control flow is the existence of two or more workflows supporting one task at the same time. Which workflow is chosen to be executed depends on some certain conditions. In previous example, because the graduating condition has been changed, so delayed seniors could not graduate unless there are two graduating approval processes, one for on-scheduled students and one for the delayed. In this situation, the time of students’ course can be used to make the decision. Figure 1 illustrates this problem more clearly.

Fig. 1. Two versions of the Graduating Approval Process: GAP1 and GAP2.

In figure 1, K98 and K99 refer to seniors starting their study in 1998 and 1999, respectively. Due to some change of mandatory courses between K98 and K99, the delayed K98s follow workflow GAP1 and on-scheduled K99s follow workflow GAP2 even though they graduate in the same year. The same problem occurs in data flows also. Although an automatic (computer-aided) decision is very convenient, a manual (human-based) decision is usually employed in the existing system. More
conveniently and practically, a commercial workflow-enabled system should provide authorized users with capability of choosing the most suitable version for a particular instance at the runtime.

Concerning data flows, the multi-version problem refers to different input or output data of each step in a workflow. They are also known as input and output messages. A message consists of many kinds of data, called data item. Any change to a message could lead to some new or existing data items to be added or removed, respectively. At that time, this workflow has more than one message version.

In addition, the bypassing problem, the second concerned exception, considers situations which can lead to some steps being skipped or substituted with other ones in the execution sequence of a particular workflow. We classify this control flow bypassing problem into two categories: conditional and manual bypassing. With the conditional bypassing problem, a step can be omitted if certain conditions are satisfied. On the other hand, the manual bypassing allows authorized users to skip a step on their own responsibility. Similarly, the bypassing problem could occur in the data flow as well, causing some input data to be omitted or replaced. We called this a message bypassing problem. Here, while some of data items in a message are missing or not applicable, the system should be able to suggest other replacements. Replacements are other predefined data items. In case all data items of the input message are replaced with empty ones, this step then is skipped and the data flow bypassing problem becomes a control flow one, as mentioned above. Unlike control flow bypassing, users should always make decision about skipping missed data items, or choosing substituted data items, or firing a data request which may trigger another workflow instance, or suspending this instance until required data is fulfilled. Figure 2 shows an example of the data flow bypassing problem in the hiring approval process of a university [15].

![Figure 2](image)

**Fig. 2.** (a) Normal input for step Validate conditions in hiring approval process and (b) a substituted input version

In the hiring approval process, normal rules require that a candidate, professor A, who is applying for a computer science lecturership should submit four items with his application: a PhD degree, curriculum vitae, teaching and researching experience proofs, and pedagogy certificate (see figure 2.a). Though missing pedagogy certificate (see figure 2.b), professor B’s application is also accepted as he has a letter from the
university’s rector, saying that his working experiences can fulfill this missing certificate.

The above discussions focus on our analysis of multi-version and bypassing problems in workflow-enabled information systems. In the next section, we will propose a database schema to support these exceptions.

4 An Exceptions-Aware Database Schema

Figure 3 shows the difference between our workflow-enabled systems’ architecture with an extensible exceptions-aware workflow database schema (b) and the previously proposed ones (a). The most significant difference in our architecture is the extensible database storing data support the upper layer to work with exceptions. Then the execution engine is extended to utilize the database. This helps our system facilitate the exception handling capability. In the application layer, add-ins mechanism is employed like previously introduced solutions (cf. section 2). However, the number of add-ins in our new system is fewer than that of the old ones because a number of their functions are done by the two lower layers.

Based on the two identified exceptions (cf. section 3), we have designed an E-R diagram which stores enough information for defining, executing, and managing workflows. Due to the limitation of the space, only the reduced version of the diagram is presented in this section.

Figure 4 shows only main entities and features of our proposed extensible exceptions-aware workflow database diagram. In this diagram, the entity Workflow represents the actual workflows. Each Workflow has at least one version, named WfVersion. A WfVersion has a set of steps which are instances of certain activities. Despite many MessageTemplates could be associated into a Step, only two messages are used as input and output. The decision is made based on their priority. The other messages are the substitutions when the main ones are missing (cf. section 3). A MessageTemplate consists of many ItemTemplates which represent a message’s data
items. Both MessageTemplate and ItemTemplate could be replaced with each other. ItemTemplate’s creator should clarify where this item comes from. This is also known as ItemSource. An ItemSource could be a Role, an Organization Unit, an Individual, or even a Step of a specific workflow.

Multi-version workflows are well-supported by the dependant relationship between Workflow and WfVersion. Each WfVersion has an effective period which helps the engine make appropriate suggestions. Users then decide which version to apply to each instance. Multi-version messages are treated as different messages. If a message M has two versions M1 and M2, any workflow using message M should have at least two versions, one for M1 and one for M2.

Fig. 4. Our proposed exception-aware E-R diagram.
For control flow bypassing, steps have an attribute, a bitwise flag combined by predefined values such as bypass-able which means that this step can be bypassed. For message bypassing, each step keeps a list of input message candidates as well as output ones, referred to as StepMessage candidates. Candidates have their own priority. When a data item or message is missing, the system lists out all remaining candidates in an ascending order of the attribute priority. This list could be extended by defining substitutions for each message and data item. Like StepMessage, a message or data item has its own substitution list which could be another message or other data items. This list is stored in three entities ItemTemplateSubstitution, MessageItemSubstitution, and MessageTemplateSubstitution.

With information stored in this fashion, the system is able to deal with multi-version and bypassing problems at the database level. In the next section, we will discuss how this E-R diagram can address the concerned problems in two examples as presented in section 3, and how to deal with other forthcoming exceptions.

5 Discussions

The first example is about multi-version of the GAP (cf. section 3). In this situation, a Workflow record stores main information of the GAP such as code, name, etc. Beside, two WfVersion records store information of the GAP1, and GAP2, which are applied for K98 and K99 seniors, respectively. Each of these rows has an attribute startDate, which is the valid time for each workflow. Based on this information and the seniors’ enrolled time, the system knows the correct GAPx applied to each senior.

Similarly, in the second example regarding hiring a lecturer, we define five data items according to the requirements. In which, we also create a record in ItemTemplateSubstitution, saying that PedagogyCertificate data item can be replaced with LetterFromRector one.

To work around other exceptions, application-level add-ins are employed (cf. figure 3). These add-ins are attached to the system at runtime. They may use an extra storage for exception handling. When an exception becomes usual, it can be further studied to find out what information should be maintained to help supporting it at the database level. The proposed database schema then can be extended to cover the new problem. With this approach, the proposed schema mitigates efforts at the application level because the developers will not have to handle the supported exceptions at database level. Obviously, our new approach is also a generalization of all known presented ones.

6 Conclusion and Future Work

Our on-going research work aims at building a flexible workflow management system. In which, the workflows’ exception handler plays a crucial role to the success. Among known exceptions, in this paper, we have focused on the two popular ones: multi-version and bypassing of workflows in SOA-based information systems. To effectively support these exceptions, we choose a database-level approach by
designing an exceptions-aware database schema that contains necessary information for exception handling. To deal with other exceptions (unsupported at the database level), like the other approaches, we choose an application-level approach by building system add-ins. These exceptions, however, will be supported at the database-level as they are well studied.

In the future, we have planned to carry out extensive experiments in order to establish the practical value of the new proposed approach. Besides, one of our great concerns is to study data exceptions in batch-processing workflows. These workflows deal with a list of records which satisfy certain conditions. However, when some of records are changed so that they no longer support the conditions, several tasks performed on these records should be roll backed and what the system should do with these records is still an open question. In addition, further investigations into other kinds of exceptions are a must and also of our great interest in the future research activities.

References

