

Making Business Process Implementations Flexible and Robust: Error Handling in the AristaFlow BPM Suite

Andreas Lanz, Manfred Reichert, and Peter Dadam

Institute of Databases and Information Systems, University of Ulm, Germany

Abstract. Process-aware information systems will be not accepted by users if rigidity or idleness due to failures comes with them. When implementing business processes based on process management technology one fundamental goal is to ensure robustness of the realized process-aware information system. Meeting this goal becomes extremely complicated if high flexibility demands need to be fulfilled. This software demonstration shows how the AristaFlow BPM Suite assists process participants in coping with errors and exceptional situations. In particular, we focus on new error handling procedures and capabilities using the flexibility provided by ad-hoc changes not shown in other context so far.

1 Introduction

During the last decade we developed the ADEPT2 next generation process management technology [1]. Due to the high interest of companies, ADEPT2 has since 2008 been transformed into an industrial-strength process management system called *AristaFlow BPM Suite* [2]. One of our basic goals is to enable robust and flexible *process-aware information systems* (PAIS) in the large scale. In particular, we want to ensure error-safe and robust process execution even at the presence of exceptions or dynamic process changes. In 2009, AristaFlow BPM Suite was applied to a variety of challenging applications in domains like healthcare, disaster management, logistics, and software engineering.

Our tool demo complements previous demos of ADEPT2 and focuses on a fundamental pillar of robust process implementations: *error handling*. One important aspect in this context is *error prevention*. We achieve the latter by applying a “correctness by construction” principle during process composition and by guaranteeing correctness and robustness in connection with dynamic process changes. This was probably the most influential challenge for our research activities. It also had significant impact on the development of the AristaFlow BPM Suite. In particular we try to detect as many potential errors as possible during buildtime (e.g. incomplete data flow specifications or deadlocks) to prevent them from happening during runtime. As we will show, however, errors cannot always be prevented. Therefore another important aspect of PAIS robustness concerns *exception handling*. Our demo will show that the AristaFlow BPM Suite provides an easy but yet powerful way to handle exceptions during runtime. In

this context ad-hoc process changes are extremely helpful. By utilizing them it becomes possible to even cope with severe process failures and to continue and complete respective processes.

The paper is structured as follows: In Section 2 we introduce a simple application scenario which we use as running example. Section 3 introduces the AristaFlow BPM Suite. In Section 4 we demonstrate how AristaFlow copes with errors that are encountered during process life and how to do this in a flexible way. Section 5 discusses related work and Section 6 concludes with a summary and outlook.

2 Application Scenario

We will use a simple example to demonstrate how errors can be handled in the AristaFlow BPM Suite. Consider Fig. 1, which shows a simple process of an online book store. In the first step a customer request is entered and required data is collected. Next the bookseller requests pricing offers from his suppliers. In this example he will request an offer from Amazon using a web service and another offer from a second company using e-mail. After he has received the pricing offers from both suppliers the bookseller checks whether he can find a special offer for the requested books in the Internet. Finally he makes an offer to his customer for the requested books.

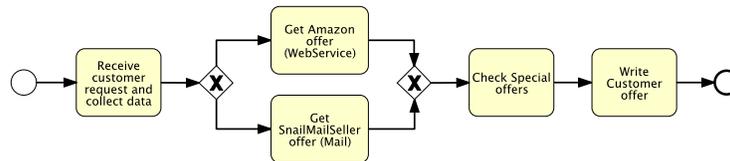


Fig. 1. Scenario: A simple process calling a web service (in BPMN notation)

As we will show, this scenario contains several sources of potential errors. Some of them can be detected and prevented at buildtime while others cannot. Assume, for example, that the process implementer does not foresee a way to enter the offer from *SnailMailSeller* into the system. In this case the final activity might fail or produce an invalid output since its input parameters are not provided as expected. Another source of errors might be the Amazon web service; e.g. it might be not available when making the request and therefore the *Get Amazon offer* activity might fail during runtime. Respective errors can be foreseen and hence considered at buildtime. However non-expected errors might occur as well; e.g., activity *Check Special offers* might fail due to troubles with the user's Internet connection.

In summary the following requirements for error-safe and robust process execution exist: On the one hand errors should be avoided during buildtime, on the other hand PAIS must enable users to effectively deal with expected and unexpected errors during runtime. In the following we show how AristaFlow BPM Suite meets these requirements.

3 AristaFlow BPM Suite

As aforementioned AristaFlow BPM Suite¹ is based on the results of the ADEPT2 project. In this research project we targeted at a process management technology which enables ease of use for process implementers, application developers and end users. Furthermore, robustness of process implementations, and support of dynamic process changes were fundamental project goals. To achieve this we realized a “correctness by construction” principle and guarantee correctness in the context of ad-hoc changes at the process instance level. Another important aspect in the context of robustness is error handling. Any PAIS will not be accepted by users if rigidity comes with it or if its use in error situations is more expensive than just handling the error by calling the right people by phone. These challenges were probably the most influential one for the whole ADEPT2 project [1, 2].

4 Demonstration of the Application Scenario

In the following we consider the scenario from Section 2 from the perspectives of the process implementer, the system, the end user, and the system supervisor. We demonstrate how each of these parties can contribute to the handling of errors.

Process implementer perspective: Fig. 2 shows a part of the process from Fig. 1 as it can be modeled using the *AristaFlow Process Template Editor*. For implementing workflows, we pursue the idea of process composition in a “plug & play” style supported by correctness checks. The latter contributes to exclude certain errors during process execution. As prerequisite, for example, implicit data flow dependencies among application services have to be made explicit to the process engine. AristaFlow provides an intuitive graphical editor and composition tool to process implementers (cf. Fig. 2), and it applies a *correctness by construction* principle by providing at any time only those operations to the user which allow to transform a structurally sound process schema into another one; i.e., change operations are enabled or disabled according to which region in the process graph is marked for applying an operation. Deficiencies not prohibited by this approach (e.g., concerning data flow) are checked on-the-fly and are reported continuously in the problem window of the *Process Template Editor*. An example can be seen in Fig. 2, where AristaFlow detects that data element *Customer price per unit* is read by activity *Write Customer offer* but not written by any preceding activity.

Generally, we should not require from process implementers that they have detailed knowledge about the internals of the application functions they can assign to process activities. However, this should not be achieved by undermining

¹ The AristaFlow BPM Suite is provided free of charge to universities for research and educational purposes. Please visit www.AristaFlow-Forum.de for more information on this topic. For commercial usage please visit www.AristaFlow.com.

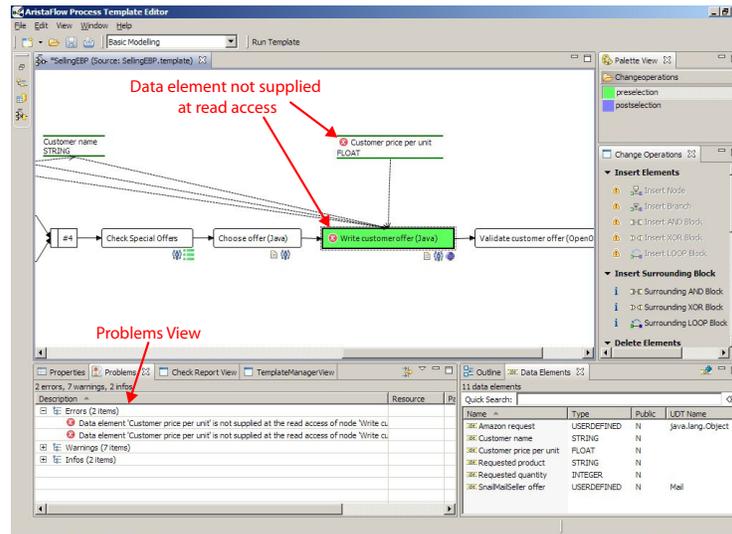


Fig. 2. AristaFlow Process Template Editor

the *correctness by construction* principle. In AristaFlow, all kinds of executables that may be associated with process activities are first registered in the *Activity Repository* as activity templates. An activity template provides all information to the *Process Template Editor*; e.g., about mandatory and optional input/output parameters or data dependencies to other activity templates. The process implementer just drags and drops an activity template from the *Activity Repository Browser* window of the *Process Template Editor* onto the desired location in the process graph.

One major advantage of this approach is that common errors, e.g. missing data bindings, can be completely prevented at buildtime. Therefore the time needed for testing and debugging can be significantly reduced; i.e., AristaFlow guarantees that process models without any detected deficiencies are sound and complete with respect to the activity templates used.

System perspective: The approach described above ensures that in principle the process model is executable by the system in an error-safe way. As always, this might not hold in practice. Again, consider the scenario from Fig. 1: the web service involved by activity *Get Amazon offer* might not be available when the process is executed, leading to an exception during runtime. Such errors can neither be detected in advance nor completely be prevented by the system.

Failures of the Amazon web service might be anticipated by the process implementer. Thus he can assign specific error handling procedures to the respective activity. Following the workflow paradigm, AristaFlow uses processes to handle exceptions, i.e., AristaFlow provides a reflective approach in which error handling is accomplished using a workflow being executed by AristaFlow. A simple error handling process is shown in Fig. 3. Depending on whether or not the failure of the process activity was triggered by the user (e.g. through an abort button) either the system supervisor is notified about the failure or

the process silently terminates. Generally, error handling processes can be arbitrarily complex, long running processes. It is important to note that AristaFlow treats error handling processes the same way as any other process. Thus they can contain any activity available in the repository. Note that this enables error handling of higher semantical level, involving users where required. If an activity fails, the respective error handling process is initiated and provided with all the data necessary to identify and handle the error, e.g. the ID of the failed activity instance, the agents responsible for the activity and the process, the cause of the error, etc. (cf. Fig. 3).

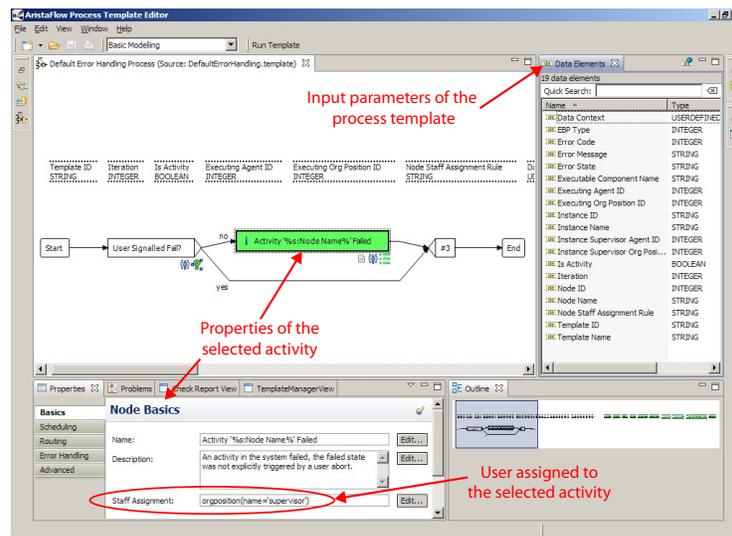


Fig. 3. A simple error handling process

After an error handling process has been created and deployed to the AristaFlow Server it can be assigned to an activity or process by simply selecting it from a list of processes. Whether or not a process is suitable as error handling process is decided based on its signature, i.e. the input and output parameters of the process.

It is also possible to assign an error handling process to a complete process instead of an activity. In this case this general error handling process will be used if no other error handling process is associated with the failing activity. In case there is no error handling process assign to either the activity or the process a system default error handling process will be used.

One of the advantages of using processes for error handling is that standard process modeling tools and techniques can be used for designing error handling strategies as well. Therefore process implementers do not need to learn any new concept to provide error handling. Another important advantage is that error handling at a higher semantical level can be easily achieved. For example, it is also possible to use more complex error handling strategies like compensation or to apply ad-hoc changes to replace parts of the failed process.

End user perspective: In certain cases the simple error handling process from Fig. 3 might be not appropriate since it increases the workload of the system supervisor. Most standard errors can also be handled in a (semi-)automatic way by the agent executing the activity. Upon failure of the respective activity the agent responsible for executing this activity could be provided with a set of possible error handling strategies he can choose from. An example for such more complex error handling process is shown in Fig. 4. Here the user can choose between a variety of ways to handle the respective error: retrying the failed process step, aborting the whole process instance or applying predefined ad-hoc changes to fix or compensate the error.

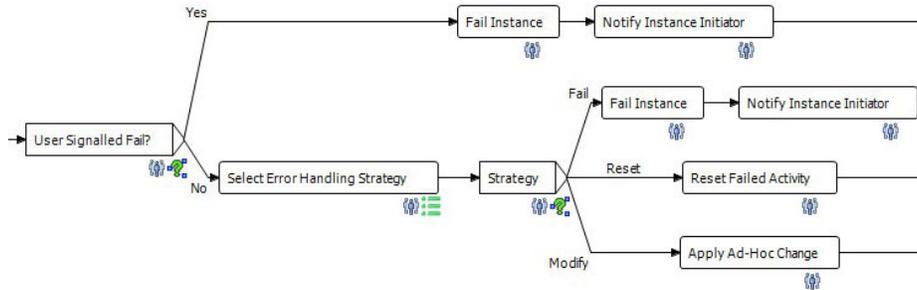


Fig. 4. A more complex error handling process involving the user

Additionally, suggested error handling strategies may depend on the background of the respective user, i.e. his knowledge and position in the organizational model and various other factors. Based on the selection of the user the respective strategy is then applied to handle the error.

Such a semi-automatic, user-centered approach offers many advantages. Since for each process activity a predefined set of possible strategies can be provided to users, they do not need to have deep insights into the process to handle errors appropriately. This allows to significantly reduce waiting times for failed instances since users can handle errors immediately by their own and do not have to wait for a probably busy helpdesk to handle errors for them. This in turn allows to relieve the helpdesk from the tedious task of handling simple process errors.

System supervisor perspective: Certain errors cannot be handled by the user. For example they might not have been foreseen at buildtime, i.e., no appropriate error handling process exists; or it might be simply not possible to handle errors in an easy and generic way. In such cases the system supervisor can use the *AristaFlow Process Monitor* shown in Fig. 5 to take a look at this process instance, to analyze its execution log, and decide for an appropriate error handling. Additionally the system supervisor can use the *AristaFlow Process Monitor* to keep track of failed instances; e.g., he may intervene if a web service becomes unavailable permanently.

Consider again our bookseller example from Fig. 1. Assume that a process instance wants to issue a request for a book using Amazon's web service facilities,

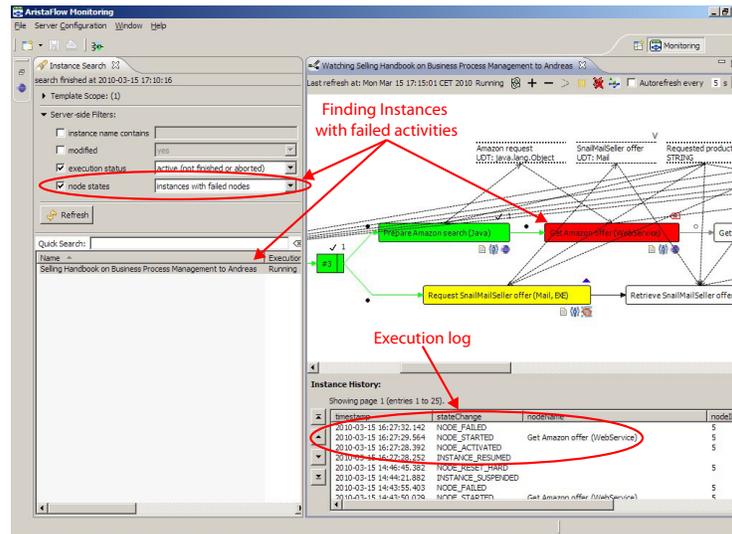


Fig. 5. Process Monitor: Monitoring Perspective

but then fails in doing so. The system administrator detects that the process is in trouble and uses the *AristaFlow Process Monitor* to take a look at this process instance (cf. Fig. 5). Analyzing the execution log of the failed activity he detects that its execution failed because the connection to Amazon could not be established. Let us assume that he considers this a temporary problem and just resets the activity so that it can be repeated once again. Being a friendly guy, he takes a short look at the process instance and its data dependencies, and sees that the result of this and the subsequent activity is only needed when executing the *Choose offer* activity. Therefore, he moves these two activities after activity *Check Special Offers*; i.e., the user can continue to work on this process instance before the PAIS tries to re-connect to Amazon (cf. Fig. 6). To accomplish this change he would switch to the *Instance Change Perspective* of the *Process Monitor* which provides the same set of change operations as the *Process Template Editor*. In fact, it is the *Process Template Editor*, but it is aware that a process instance has been loaded and, therefore, all instance-related state information is taken additionally into account when enabling/disabling change operations and applying correctness checks. The system administrator would now move the two nodes to their new position by using the respective standard change operation. The resulting process is depicted in Fig. 6. Assume now that the web service problem lasts longer than expected and, therefore, the user wants to call Amazon by phone to get the price that way. In this case he would ask the system administrator to delete the activities in trouble and to replace them with a form-based activity which allows to enter the price manually.

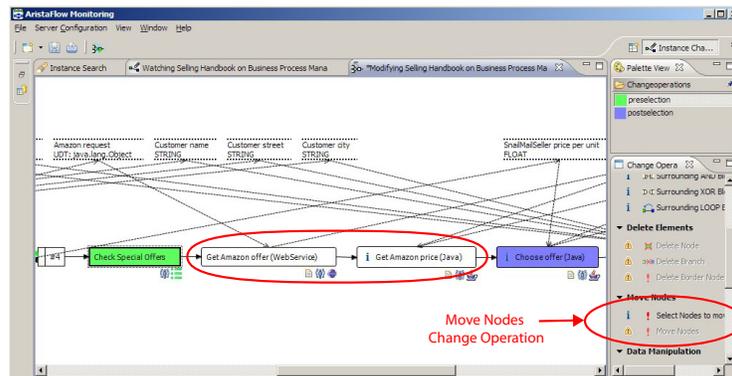


Fig. 6. Process Monitor: Instance Change Perspective

5 Related Work

Besides ADEPT, YAWL [3] has been one of the first workflows engines to support some sort of “correctness by construction” as well as correctness checks at buildtime. jBPM [4] rudimentarily supports ad-hoc deviations of running process instances, but without any correctness assurance as provided by the AristaFlow BPM Suite. Most BPEL-based workflow engines like Websphere Process Server [5] support error handling processes using fault handlers, but without the possibility to structurally change process instances during runtime.

6 Summary and Outlook

Due to its “correctness by construction” principle and its comprehensive support of ad-hoc changes during runtime, as well as the possibility to define arbitrary error handling processes, AristaFlow is well suited to enable robust process implementations while preserving the possibility to flexibly react to exceptional situations during runtime. Currently, we investigate the handling of other kinds of errors (e.g. time related errors).

References

1. Reichert, M., Rinderle-Ma, S., Dadam, P.: Flexibility in process-aware information systems. *LNCS Transactions on Petri Nets and Other Models of Concurrency (ToPNoC)* **2** (2009) 115–135
2. Dadam, P., Reichert, M.: The ADEPT project: A decade of research and development for robust and flexible process support - challenges and achievements. *Computer Science - Research and Development* **22** (2009) 81–97
3. Russell, N., ter Hofstede, A.H.M.: Surmounting BPM challenges: the YAWL story. *Computer Science - R&D* **23** (2009) 67–79
4. Koenig, J.: JBoss jBPM (whitepaper) (2004)
5. Kloppmann, M., Konig, D., Leymann, F., Pfau, G., Roller, D.: Business process choreography in websphere: Combining the power of BPEL and J2EE. *IBM Systems Journal* **43** (2004) 270–296