Robust and Flexible Error Handling in the AristaFlow BPM Suite

Andreas Lanz, Manfred Reichert, and Peter Dadam

Institute of Databases and Information Systems, University of Ulm, Germany
{Andreas.Lanz,Manfred.Reichert,Peter.Dadam}@uni-ulm.de

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 resulting process-aware information system. Meeting this goal becomes extremely complicated if high flexibility demands need to be fulfilled. This paper shows how the AristaFlow BPM Suite assists process participants in coping with errors and exceptional situations in a flexible and robust way. In particular, we focus on novel error handling procedures and capabilities using the flexibility provided by ad-hoc changes not shown in other context so far.

Keywords: Process-aware Information System, Adaptive Process, Error Handling.

1 Introduction

During the last decade we developed the next generation process management system ADEPT2 [1]. Due to the high interest of companies in this technology, we transferred it into an industrial-strength process management system called *AristaFlow BPM Suite* [2]. One of the fundamental goals of AristaFlow is to enable robust and flexible *process-aware information systems* (PAISs) in the large scale. In particular, we want to ensure error-safe and robust process execution even at the occurrence of exceptions or dynamic process changes. Recently, ADEPT2 and AristaFlow, respectively, were applied to a variety of challenging applications in domains like healthcare [3,4], disaster management [5], logistics [6], and software engineering [7]. The overall goal was to learn more about process flexibility issues in advanced applications and to study how adaptive process management technology can be applied to deal with errors and exceptions.

This paper complements our previous work on ADEPT2 [1,2,8,9] and focuses on a fundamental pillar of any robust process implementation: *error handling*. One important dimension in this context concerns *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 as well. This was probably the most influential challenge for our whole research. It also had significant impact on the development of

P. Soffer and E. Proper (Eds.): CAiSE Forum 2010, LNBIP 72, pp. 174–189, 2010.

the AristaFlow BPM Suite. In particular we try to detect as many errors as possible (e.g. incomplete data flow specifications or deadlocks) already at buildtime in order to obviate their occurrence during runtime. In general errors cannot be always foreseen and thus be prevented. Therefore, another important dimension of PAIS robustness concerns *exception handling*. We will show that AristaFlow provides an easy, but yet powerful tool to handle exceptions during runtime. As we will show, in respect to flexible exception handling ad-hoc process changes have proven to be extremely useful. By utilizing them it even becomes possible to cope with severe process failures and to continue repaired processes in a correct way.

In Section 2 we introduce a simple application scenario which we use as illustrating example throughout the paper. Section 3 gives backgrounds on the AristaFlow BPM Suite. In Section 4 we show how this process management system copes with errors that might occur during process execution, and how to do this in a flexible and robust way. Section 5 summarizes real-world cases to which AristaFlow was applied. Section 6 discusses related work and Section 7 concludes with a summary and outlook.

2 Illustrating Application Scenario

We use a simple example to illustrate how different kinds of errors within PAISs can be handled when using AristaFlow. Consider Fig. 1. It 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 vendor using e-mail. After having received the pricing offers from both suppliers, the bookseller checks whether or not 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.

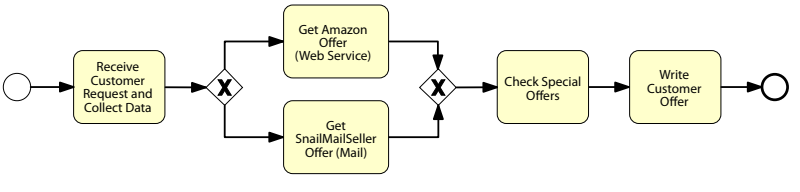

Fig. 1. Scenario: A simple process calling a web service (in BPMN notation)

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, activity *Write Customer Offer* might fail or produce an invalid output since its input parameters are not provided as expected. Another source of error is the Amazon web service,

Table 1. Typical errors in a PAIS

Buildtime	**Runtime**
– structural errors, e.g. • deadlocks • isolated activities – dataflow errors, e.g. • missing parameter values	– activity failure, e.g. • broken database connection • invalid input – context failure, e.g. • mismatch between real-world and process running in PAIS

which it might be not available when making the request and therefore the activity *Get Amazon Offer* might fail during runtime. Respective errors can be foreseen and hence be considered at buildtime. However, non-expected errors might occur as well; e.g., activity *Check Special Offers* might fail due to troubles with the Internet connection. Table 1 lists some typical errors in PAISs. For a more complete discussion we refer interested readers to [10].

In summary the following requirements for error-safe and robust process execution exist:

- Errors should be obviated at buildtime if possible.
- Users should be enabled to effectively deal with both expected and unexpected errors during runtime.
- Error and exception handling must not counteract formal process properties (e.g., proper termination) as guaranteed at buildtime by applying the aforementioned "correctness-by-construction" principle.

3 Background

As aforementioned the AristaFlow BPM Suite[1] is based on research results we obtained in the ADEPT1 and ADEPT2 projects. With ADEPT1 a first powerful prototype of the ADEPT technology became operational [11]. Based on hands-on experiences we gathered in several projects in the healthcare domain its most interesting feature was certainly the support of ad-hoc deviations [12]. Later ADEPT1 served as implementation platform for numerous projects (e.g., [4,6,13]). From this, we learned that a better integration of the offered features as well as an open API was needed. In 2004 we therefore started the development of the ADEPT2 project in which we targeted at a process management technology which enables ease of use for all user groups involved in the specification and execution of processes. Furthermore, the realization of robust process implementations and flexible support of dynamic process changes were fundamental project goals. Ensuring both robustness and ease of use for *process implementers* and *application developers* are indispensable in this context. This challenge was

[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.

probably the most influential one for the whole project [1,2]. It had significant impact on the development of the used process meta model as well as on our work on process flexibility and adaptivity. It meant, in essence, the following:

1. We have to hide the inherent complexity of process-orientation (especially in conjunction with flexibility) as far as possible from *system supervisors* and *application developers*; i.e., we have to perform all complex things "beneath the surface" in the process management system.
2. We have to provide powerful, high-level interfaces to *application developers*, based on which they can implement easily usable end user interfaces.

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. "Correctness-by-construction" is realized by providing a theory [11] which precisely defines correctness criteria for the ADEPT meta model (e.g., absence of deadlocks, no isolated activities). This theory also defines a comprehensive set of change operations with pre-/post-conditions which ensure that, if the desired change satisfies the preconditions, the resulting process schema will be correct again, i.e., change operations allow to transform a structurally correct process schema into another structurally correct one [11]. Additionally, all change operations obey that data flow correctness is not violated. This is achieved by utilizing the block-structure of the underlying process meta model [11]. Particularly, a process schema can only be deployed to the process engine if it satisfies the above mentioned correctness criteria [2]. Finally, the change operations allow to safely deviate from the predefined process schema during runtime.

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. Therefore, users may deviate from the pre-modeled process by structurally adapting it, but without violating correctness properties.

4 Supporting the Described Application Scenario

In the following we re-consider the scenario from Section 2 from the perspectives of the *process implementer*, the *system*, the *end user*, the *system supervisor*, and the *process reengineer*. We demonstrate how each of these parties can be involved in handling errors. This is by no means a complete list of all parties which participate in the business process life cycle. We focus on those user groups involved in the specification and execution of processes, but exclude other user groups (e.g. business analysts or business process owners) who are mainly engaged in the monitoring and analysis of business processes.

4.1 Process Implementer Perspective

We first consider the *process implementer*. He is responsible for correctly modeling processes as well as for linking their activities to application services.

Process Modeling. Fig. 2 shows a part of the process from Fig. 1 as it can be modeled using the *AristaFlow Process Template Editor*. Additionally, it depicts parts of the data flow between the process activities (e.g., data element *Customer Name* is read by activity *Write Customer Offer (JAVA)*). For implementing processes, we pursue the idea of process composition in a *"plug & play"* style which is additionally supported by on-the-fly correctness checks if needed. More precisely, AristaFlow provides an intuitive graphical editor and composition tool to *process implementers* (cf. Fig. 2), and it applies the *correctness-by-construction* principle by providing at any time only those operations to the user which allow to transform one structurally sound process schema into another one; i.e., change operations are enabled or disabled according to the region which is marked in the process graph for applying an operation [2]. Deficiencies not prohibited by this approach (e.g., concerning data flow correctness) are checked on-the-fly and are reported continuously in the problem window of the *Process Template Editor*. As a prerequisite, for example, implicit data flow dependencies among the application services (implementing the process activities) have to be made known to the process engine. An example is depicted in Fig. 2, where AristaFlow detects that data element *Customer Price per Unit* is read by activity *Write Customer Offer* although it is not written by any preceding activity. Such deficiencies are highlighted to the *process implementer* who then can correct the model as required.

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

Activity Implementation and Configuration. In AristaFlow, all kinds of executables (e.g., user forms, web services, database components, file operations,

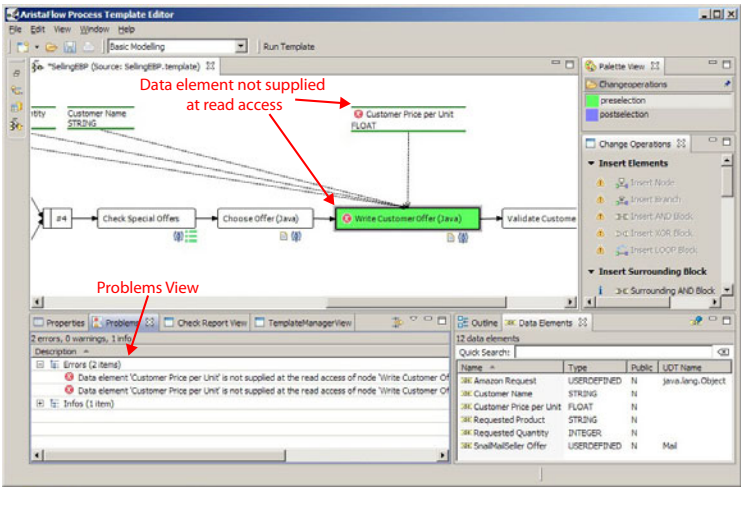

Fig. 2. AristaFlow Process Template Editor

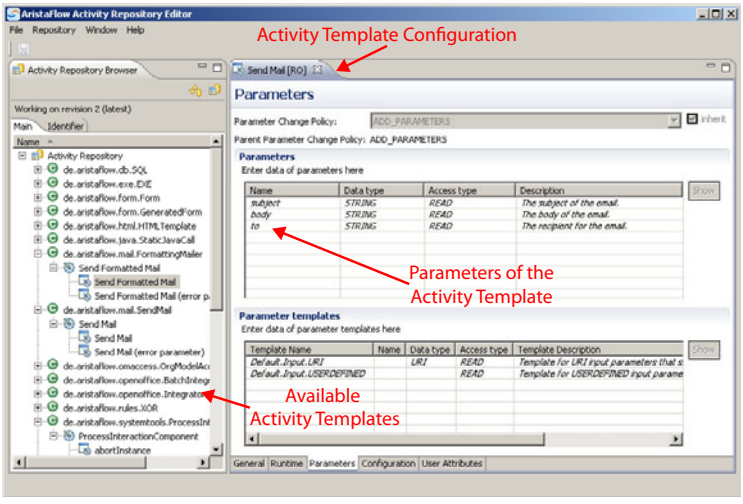

Fig. 3. Activity template configuration with AristaFlow Activity Repository Editor

etc.) that may be associated with process activities first need to be registered in the *Activity Repository* as activity templates (cf. Fig 3). An activity template, in turn, provides required information to the *Process Template Editor*; e.g., about mandatory and optional input/output parameters or about data dependencies to other activity templates. An *application developer* who wants to introduce a new application service to the PAIS must first implement a corresponding activity template and then add it to the *AristaFlow Activity Repository*. This way it becomes available and accessible within the *Process Template Editor* during process composition.

To ease the implementation of activity templates, AristaFlow provides several levels of abstraction. This includes the execution environment at the lowest one. An execution environment defines the set of methods (e.g., initialization and execution of the activity) needed to interact with the AristaFlow runtime system (i.e., the process engine) as well as to implement the operations and properties (e.g., shall the activity be suspendable, abortable, etc.) to be provided by the activity template. However, the implementation of such execution environment requires knowledge about AristaFlow internals and, therefore, will typically not be the task of an ordinary *application developer*, but be performed by *system implementers*. Next, activity templates provide predefined configuration sets for execution environments. Depending on the intended purpose of usage, an activity template can be very specific or rather generic, where a more specific activity template can be derived from a generic one. An activity template for a web service call, for example, may be completely pre-configured; e.g., the input/output parameters and all configuration and connection settings are fixed. In this case, the only remaining task for the *process implementer* is to check whether or not the proposed mapping of input/output parameters to process data elements (i.e., the process variables used within the respective process to exchange data

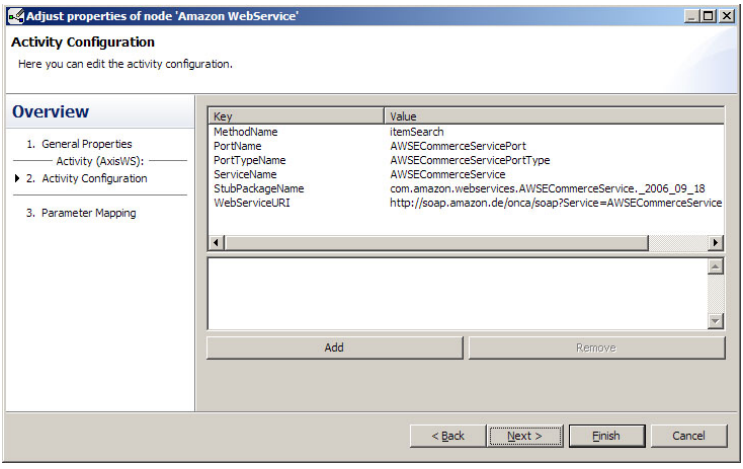

Fig. 4. Activity configuration of a generic web service activity template

among activities) is correct. A more generic web service activity template, in turn, may allow the *process implementer* to specify connection details of the web service (as illustrated in Fig. 4) or even allow to configure the number and types of input/output parameters.

In general, the AristaFlow runtime environment requires certain information about the runtime behavior of the activities; e.g., whether or not they may be aborted, suspended or undone. The implementer of an activity template has to inform the AristaFlow runtime environment which of these facilities are supported by the activity. For this case, he must also provide the implementation of this functionality in the respective execution environment [8]. Such provision of activity templates at different levels of abstractions constitutes a powerful and flexible means for process composition in a "plug & play"-like fashion.

Given the *AristaFlow Process Template Editor* and the *AristaFlow Activity Repository*, the *process implementer* just drags and drops the activity templates from the *Activity Repository Browser* window of the *Process Template Editor* onto the desired location in the process graph. Configuration efforts in respect to the chosen activity template are then reduced to the provision of the remaining configuration values in the configuration wizard of the respective activity template (cf. Fig. 4). For example, if a web service activity template shall be implemented one page of the wizard will fix the required input parameters and the output parameters for the attribute values of the web service, a second one the settings of binding information for the web service (e.g. the web service URL) (cf. Fig. 4), and a third one the mapping of activity input/output parameters to process data elements (cf. Fig. 5).

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

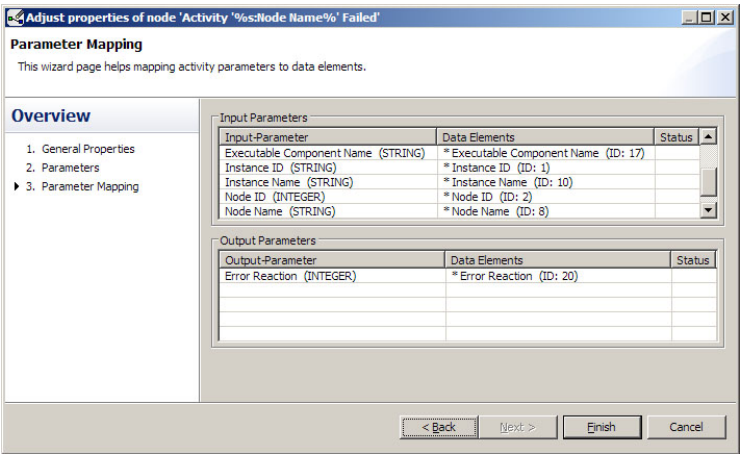

Fig. 5. Mapping activity parameters to data elements

guarantees that released process implementations are sound and complete with respect to the data dependencies of the used activity templates.

Nevertheless, *correctness-by-construction* and automated checks can only ensure correct execution of processes on a syntactical level. By contrast, semantic errors cannot be detected by automated checks. Therefore, AristaFlow provides a sophisticated *Test Environment* which allows *process implementers* to test processes prior their release in the production system. Using the *AristaFlow Test Client* it is even possible to execute only partially specified processes, i.e., not all activities need to have associated activity templates or staff assignment rules may be still undefined. In case an activity has no assigned activity template, in this test mode the user will be supported by automatically generated forms, which allow him to review the input parameters of the activity and to set its output parameters. In particular, this enables *process implementers* to get rapid feedback from future users during process development. Consequently, semantic errors and misinterpretations can be partially detected at a very early stage of the process implementation phase.

4.2 System Perspective

In principle, the approach described in Section 4.1 ensures that released process models are 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 associated with activity *Get Amazon Offer* might not be available during process execution, leading to an exception that needs to be handled. Such errors can neither be detected in advance nor be prevented by buildtime checks.

However, 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 a strict process paradigm, AristaFlow runs specific processes to handle exceptions; i.e., we provide a reflective approach

in which error handling itself can be accomplished base on a (normal) process instance running in the PAIS. A simple error handling process is shown in Fig. 6. 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 terminates silently. Generally, error handling processes can be arbitrarily complex and long running processes (e.g., comprising compensation tasks). It is noteworthy that AristaFlow treats error handling processes the same way as any other process. Thus they may refer to any activity registered in the repository. In particular, this enables error handling at a higher semantic level, as well as the involvement of users if required. We further can assign error handling processes to the activities of another error handling process if desired. This way it becomes possible to implement several layers of error-handling on top of each other.

If an activity fails the error handling process assigned to it will be initiated and be provided with all data necessary to identify, classify, and handle the error; e.g. the ID of the failed activity instance, the agents (i.e., user or automated agent) responsible for the activity, and the cause of the error (cf. Fig. 6).

After an error handling process has been created and deployed to the AristaFlow Server, it can be assigned to an activity by simply selecting it from a list of available processes. It is further possible to assign an error handling process to the whole process instead of single activities. This general error handling process will then be used if a failed activity has no directly associated error handling process. In case there is no error handling process being assigned to either the activity or the process a default error handling process will be used.

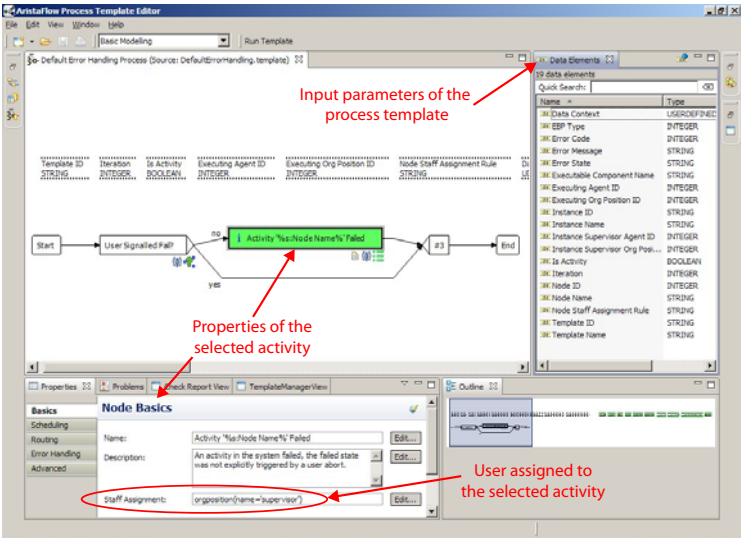

Fig. 6. A simple error handling process

Another advantage of user-defined processes for error handling is the possibility to use standard process modeling tools and techniques for designing error handling strategies. Therefore *process implementers* do not need to learn any new concept to enable error handling. Another important advantage is that error handling at a higher semantic 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.

4.3 End User Perspective

In certain cases simple error handling processes like the one depicted in Fig. 6 might be not appropriate since they increase 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 its execution could be provided with a set of possible error handling strategies he can choose from. An example for a more complex error handling process is depicted in Fig. 7. Here the agent can choose between several ways to handle the occurred error: retrying the failed process step, aborting the whole process instance, or applying predefined changes to fix or compensate the error. Additional error handling strategies may, for example, include the escalation of the respective case to a responsible supervisor or an enquiry with a more advanced user on how to handle the respective error.

Generally the concrete error handling strategies suggested to particular users may depend on their capabilities and position as captured in the organizational model. Consequently the assignment of error handling processes and respective activities can be done dynamically and user-dependent in AristaFlow.

To flexibly cope with errors and exceptions, end users are not only allowed to dynamically adapt process instances, but are also assisted in retrieving and reusing knowledge about previously performed process changes applied in similar problem context. Basic to this change reuse is the integration of the adaptive process management system with concepts and methods provided by case-based reasoning technology. This allows for expressing the semantics of process changes, for memorizing these adaptations, and for reusing them in similar context later. We implemented such an integrated approach in the ProCycle project [14].

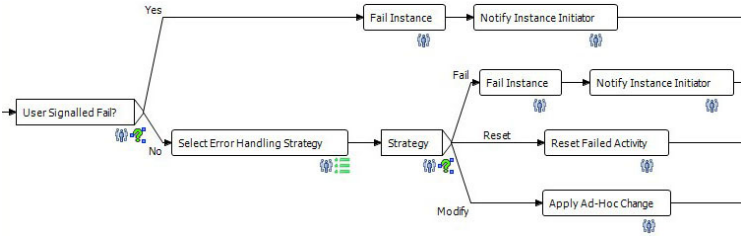

Fig. 7. A more complex error handling process involving the user

The described semi-automatic, user-centered approach offers many advantages. Since for each process activity a predefined set of possible error handling strategies can be provided to users, they do not need to have detailed knowledge about the process to handle errors appropriately, but are guided by the system in error situations instead. This particularly fosters the reduction of waiting times in the context of failed activity instances since users can handle errors immediately by their own and do not have to wait for their busy help desk to do this for them.

4.4 System Supervisor Perspective

Certain errors cannot be handled by the user. This applies, for example, to errors that might not have been foreseen at buildtime, i.e., no appropriate error handling process exists. In other cases it might be simply not possible to handle errors in an easy and generic way. At that point a *system supervisor* should be notified about the error. For example, this can either be done through an automatic notification by the error handling process (e.g., by adding a respective item to his worklist) or by a user calling the help desk by phone. The *system supervisor* then can use the *AristaFlow Process Monitor* shown in Fig. 8 to identify the process in trouble. This can be done by either using the process identifier provided by the error process or by applying different sets of filters to the list of process instances currently known by the system (cf. Fig. 8). Process instances can, for example, be identified by searching for active instances with failed activities, searching for modified instances, or searching for instances by name. Next, the *system supervisor* can take a look at the process instance in trouble, analyze its execution log, and decide for appropriate error handling measures. Additionally, the *system supervisor* can use the above described filters of the *AristaFlow Process Monitor* to keep track of failed instances; e.g., he can intervene if a web service becomes unavailable for a longer period of time.

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, but then fails in doing so. The *system supervisor* detects that the process is in trouble and uses the *AristaFlow Process Monitor* to take a look at this process instance (cf. Fig. 8). 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 as 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 realizes that the results of this and the subsequent activity are 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. 9). 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* (for a general overview on process change patterns see [15]). In fact, the *Instance Change Perspective* is the *Process Template Ed-*

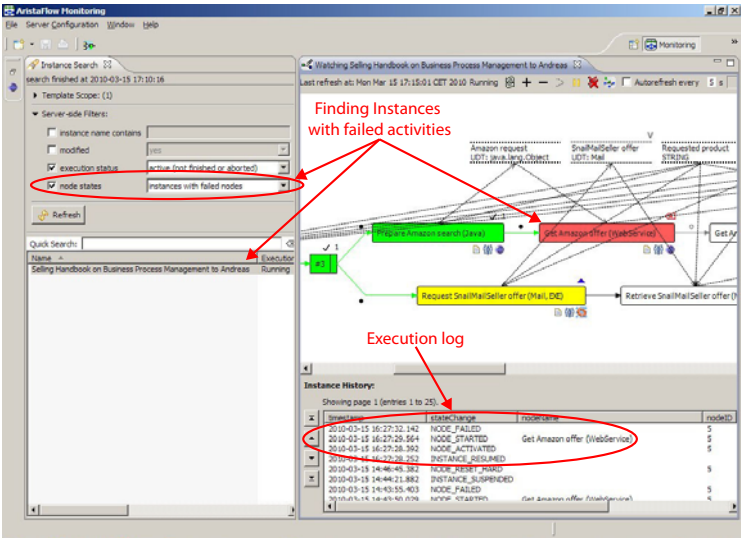

Fig. 8. Process Monitor: Monitoring Perspective

itor, 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.[2] 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. 9.

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 supervisor* to delete the activities in trouble and to replace them with a form-based activity (or any other suitable activity) which allows to enter the price manually. Note that structural ad-hoc changes provide the means to realize such advance exception handling policies.

4.5 Process Reengineer Perspective

As discussed, in AristaFlow certain errors can be handled by dynamically adapting the corresponding process instance. When considering a larger instance collection, respective model adaptations often result in a large number of model variants derived from the same process model, but slightly differing in structure. We foster learning from process instance adaptations in order to discover an improved process model that can serve as reference for future process instances of the respective type. An algorithm enabling such learning and model improvement is presented in [16]. Finally, after identifying potential changes the process (re-)engineer can perform a schema evolution. In this context he may also migrate running instances to the new process model version if desired [9].

[2] Whether or not a particular change can be applied in the current process state is decided based on well defined correctness criteria as suggested by ADEPT2 [9].

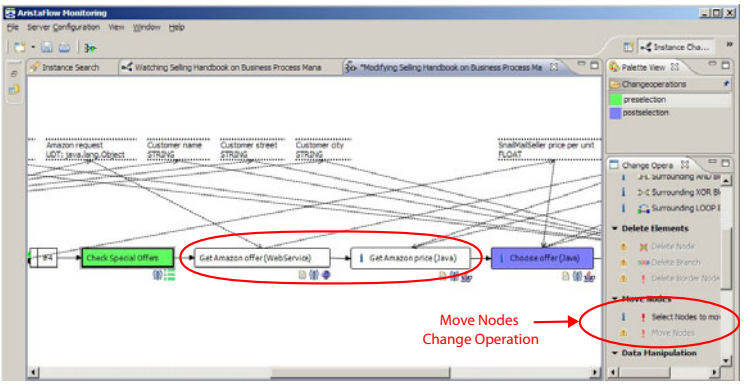

Fig. 9. Process Monitor: Instance Change Perspective

5 Applying AristaFlow in Practice

Recently, several projects applied the AristaFlow process management technology in challenging domains like healthcare, logistics, disaster management, and software engineering. In each project sophisticated PAISs were realized which make use of the AristaFlow error handling features. For us, one important goal of these projects was to understand whether or not the designed features are applicable in practice.

Applying AristaFlow to Software Engineering Processes. The Q-Advice project [7] tries to assist overburdened software engineers by providing orientation and guidance through automated workflows. Yet, since there are so many different kinds of issues with ambiguous and subjective delineation, it is difficult and burdensome to universally and correctly model them in advance. This also leads to workflows of considerable size and complexity. The Q-Advice project tries to alleviate this by starting with a basic and simple workflow for each case and then, utilizing context information, dynamically extends it with activities matching the current situation. Overall, Q-Advice provides situational and tailored support and guidance for software engineers. In particular the workflows resulting from the Q-Advice approach are much simpler than pre-modeled workflows would be. Respective adaptation features directly make use of AristaFlow's change facilities. In this context, the AristaFlow error handling processes and the provided change support features have proven to be especially useful. For example, if a bug is detected during the final steps of a release phase it needs to be decided whether or not this bug shall to be fixed prior to the release. If the bug turns out to be a show stopper, in turn, it may become necessary to change great parts of the release process.

Applying AristaFlow in Healthcare and Logistics. Healthcare and logistics are both characterized by high flexibility demands. Additionally, both require tools that are easy to use since domain specialists have no IT knowledge. By supporting domain-specific views on processes (e.g., clinical pathways) and services, the *SPOT* project [3] (*Service-based technologies for orchestrating*

PrOcesses in logisTics and healthcare) enables end-users to actively shape the different phases of the process life cycle. In both domains exceptional situations are part of the daily business and need to be handled quickly and in a way easy to use by end-users. Another requirement fully met by AristaFlow concerned application integration, i.e., to integrate heterogeneous, autonomous applications in a process-oriented way.

Applying AristaFlow in Disaster Management. The project on *process-aware, cooperative emergency management of water infrastructures* [5] aimed at improving and supporting emergency management for flood responses through new IT methods. During the project, procedures and courses of actions were analyzed, and results were mapped to formal process models. On the basis of an organizational model, the activities of the process models were assigned to responsible parties, thus enabling the involved organizations to act faster and in a more coordinated way. AristaFlow was used to manage and control the procedures and tasks during flood events as well as the corresponding information flow. Thus, it supported responders in planning and executing flood response operations in a coordinated, but flexible way. As emergency situations are not predictable, one important aspect was to provide the necessary flexibility, while ensuring robustness and error-safety of the PAIS.

Overall, in all these projects, adaptive process management technology for dynamically defining, composing and adapting process instances as well as for flexibly handling errors was indispensable. AristaFlow was one of the few systems that offers adaptation interfaces as well as adaptation features for this. All mentioned projects have proven that the previously described concepts are highly necessary in practice. At buildtime they enable us to detect and obviate design and runtime errors. This allows for easier and faster development of processes as the time needed for testing can be significantly reduced. At runtime the provided adaptation features enable PAIS utilizing the AristaFlow system to rapidly and dynamically react to exceptional situations like errors or changes in a process execution context. Additionally, the flexibility provided by the AristaFlow system allows processes to be at first only partially specified and then be further developed as they are executed.

6 Related Work

Besides ADEPT, YAWL [17] has been one of the first process engines to support some sort of "correctness-by-construction" as well as correctness checks at buildtime. jBPM [18] rudimentarily supports ad-hoc deviations of running process instances, but without any correctness assurance as provided by the AristaFlow BPM Suite. Furthermore, only simple adaptation patterns are provided [15]. Most BPEL-based workflow engines like WebSphere Process Server [19] support error handling processes using fault handlers, but without the possibility to structurally change process instances during runtime.

Declarative workflow systems like Declare [20] and Alaska Simulator [21] allow for a great degree of flexibility during process execution, but mostly lack means for ensuring robust process execution and for enabling application integration.

Additionally, they neither support the specification of data dependencies between activities nor correctness checks of the respective data flow.

Exception handling is an important topic in PAISs. In [10] the authors propose a classification framework for workflow exception handling in terms of workflow exception patterns. [22] incorporates language primitives for error handling into workflow systems and, like AristaFlow, allows error handling strategies to be modeled in the same notation as used for workflow processes. [23] uses a case-base reasoning approach to match errors with suitable error handling strategies. In [24] the authors propose several algorithms for mining process execution logs regarding exception handling. Based on the way past exceptions were handled, proposals are made on how to cope with the current one.

7 Summary and Outlook

Most existing PAISs are ill-equipped to meet the requirements of complex, real-world processes especially in the context of exceptional situations. Although a lot of papers claim that respective problems are solved in principle, we strongly believe that up to now they are even not completely understood. As shown, our tool provides an integrated solution for the easy and flexible handling of a variety of errors in exceptional situations. Most of the discussed projects would not have been possible without flexible process support as provided by the AristaFlow BPM Suite. 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.

The projects also showed that the provided flexibility is difficult to control by end-users, i.e., there is need for further research. Especially more sophisticated user-interfaces and user assistance are required for daily work. In this context, other perspectives like temporal constraints [25] become increasingly important.

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, 115–135 (2009)
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, 81–97 (2009)
3. Frauenhofer ISST: SPOT Project (2010), `http://www.spot.fraunhofer.de/` (accessed 07.09.2010)
4. Müller, R., Rahm, E.: Dealing with logical failures for collaborating workflows. In: Scheuermann, P., Etzion, O. (eds.) CoopIS 2000. LNCS, vol. 1901, pp. 210–223. Springer, Heidelberg (2000)
5. Wagenknecht, A., Rüppel, U.: Improving resource management in flood response with process models and web GIS. In: TIEMS 2009, pp. 141–151 (2009)
6. Bassil, S., Keller, R., Kropf, P.: A workflow-oriented system architecture for the management of container transportation. In: Desel, J., Pernici, B., Weske, M. (eds.) BPM 2004. LNCS, vol. 3080, pp. 116–131. Springer, Heidelberg (2004)

7. Grambow, G., Oberhauser, R., Reichert, M.: Semantic workflow adaption in support of workflow diversity. In: 4th International Conference on Advances in Semantic Processing, SEMAPRO 2010 (2010)
8. Reichert, M., Dadam, P., Jurisch, M., Kreher, U., Göser, K., Lauer, M.: Architectural design of flexible process management technology. In: Proc. PRIMIUM Subconference at MKWI 2008, pp. 415–422 (2008)
9. Rinderle, S., Reichert, M., Dadam, P.: Flexible support of team processes by adaptive workflow systems. Distributed and Parallel Databases 16, 91–116 (2004)
10. Russell, N., van der Aalst, W., ter Hofstede, A.: Workflow exception patterns. In: Martinez, F.H., Pohl, K. (eds.) CAiSE 2006. LNCS, vol. 4001, pp. 288–302. Springer, Heidelberg (2006)
11. Reichert, M., Dadam, P.: ADEPTflex - supporting dynamic changes of workflows without losing control. Journal of Intelligent Information Systems 10, 93–129 (1998)
12. Reichert, M., Hensinger, C., Dadam, P.: Supporting adaptive workflows in advanced application environments. In: Proc. EDBT Workshop on Workflow Management Systems, pp. 100–109 (1998)
13. Bassil, S., Benyoucef, M., Keller, R., Kropf, P.: Addressing dynamism in e-negotiations by workflow management systems. In: Proc. 13th International Workshop on Database and Expert Systems Applications (DEXA 2002), pp. 655–659 (2002)
14. Weber, B., Reichert, M., Wild, W., Rinderle-Ma, S.: Providing integrated life cycle support in process-aware information systems. Int. Journal on Cooperative Information Systems 18, 115–165 (2009)
15. Weber, B., Reichert, M., Rinderle-Ma, S.: Change patterns and change support features - enhancing flexibility in process-aware information systems. Data and Knowledge Engineering 66, 438–466 (2008)
16. Li, C., Reichert, M., Wombacher, A.: Discovering reference models by mining process variants using a heuristic approach. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, pp. 344–362. Springer, Heidelberg (2009)
17. Russell, N., ter Hofstede, A.: Surmounting BPM challenges: the YAWL story. Computer Science - Research and Development 23, 67–79 (2009)
18. Koenig, J.: JBoss jBPM (whitepaper) (2004)
19. Kloppmann, M., König, D., Leymann, F., Pfau, G., Roller, D.: Business process choreography in WebSphere: Combining the power of BPEL and J2EE. IBM Systems Journal 43, 270–296 (2004)
20. van der Aalst, W., Pesic, M., Schonenberg, H.: Declarative workflows: Balancing between flexibility and support. Computer Science - Research and Development 23, 99–113 (2009)
21. Weber, B., Reijers, H.A., Zugal, S., Wild, W.: The declarative approach to business process execution: An empirical test. In: van Eck, P., Gordijn, J., Wieringa, R. (eds.) CAiSE 2009. LNCS, vol. 5565, pp. 470–485. Springer, Heidelberg (2009)
22. Hagen, C., Alonso, G.: Exception handling in workflow management systems. IEEE Transactions on Software Engineering 26, 943–958 (2000)
23. Luo, Z., Sheth, A., Kochut, K., Miller, J.: Exception handling in workflow systems. Applied Intelligence 13, 125–147 (2000)
24. Hwang, S., Ho, S., Tang, J.: Mining exception instances to facilitate workflow exception handling. In: Proc. Database Systems for Advanced Applications, pp. 45–52 (1999)
25. Lanz, A., Weber, B., Reichert, M.: Workflow time patterns for process-aware information systems. In: Nurcan, S., Ukor, R. (eds.) BPMDS 2010 and EMMSAD 2010. LNBIP, vol. 50, pp. 94–107. Springer, Heidelberg (2010)