Enabling Flexible and Robust Business Process Automation for the Agile Enterprise

Manfred Reichert

Abstract During the last decade process-aware information systems (PAISs) have become increasingly popular to digitize business processes and to effectively support them at the operational level. In many application domains, however, PAISs will not be accepted by users if rigidity comes with them. Ensuring PAIS robustness, in turn, becomes extremely complicated if high flexibility demands need to be fulfilled. To cope with the dynamic nature of business processes, we developed AristaFlow, a next generation process management technology that enables comprehensive process lifecycle support. In addition to standard process management services, AristaFlow can handle exceptions, change the execution of running business cases on the fly, efficiently deal with uncertainty, and support the evolution of business processes over time. This paper discusses how AristaFlow assists the various stakeholders of a PAIS to cope with errors and exceptional situations, while still meeting robustness needs. In particular, we focus on new error handling procedures and capabilities utilizing the flexibility provided by ad-hoc changes.

1 Introduction

In today's dynamic business world, the success of an enterprise increasingly depends on its ability to react to environmental changes in a quick and flexible way. Examples of changes include regulatory adaptations (e.g., Sarbanes-Oxley), market evolution, changes in customer behavior, redesigned business processes, and strategic shifts. Therefore, enterprises have identified *business agility* as a competitive advantage to address business needs like increasing product variability or faster time-to-market as well as to tightly align business and IT. Improving the efficiency and quality of their *business processes* and optimizing their interactions with partners and customers have become crucial success factors for enterprises [15, 21].

Manfred Reichert, Ulm University, e-mail: manfred.reichert@uni-ulm.de

Contemporary enterprise information systems, which are often organized in a data- or function-centric way, lack *process awareness* hindering business agility. In many cases, enterprises prefer abandonning new business initiatives rather than attempting to adapt their enterprise software. To better support their business processes and to manage them in a more flexible manner, however, enterprises are increasingly interested in aligning their information systems in a process-centric way offering the right *business functions* to the right *users* at the right *point in time* along with the needed *information* and *application services* [25]. Along this trend, a new generation of enterprise information systems–so-called *process-aware information systems (PAISs)*–has emerged [21], which aim to overcome this inflexibility.

Examples of PAISs include workflow management systems, case handling tools, and service orchestration engines [25]. In spite of several success stories on the uptake of PAISs, the latter have not been widely adopted in industry yet [11]. A major reason for their low use is the rigidity enforced by them, which inhibits the ability of enterprises to respond to process changes or exceptions in an agile way [22]. When efforts are taken to improve and automate the flow of business processes, however, in many domains (e.g., healthcare) it is crucial not to restrict staff [18, 13]. For example, first attempts to change the function- and data-centric views on patient treatment processes in hospitals failed whenever rigidity came with them [13, 16]. Variations in the course of a treatment process are inherent to medicine, and to some degree the unforeseen event constitutes a "normal" phenomenon [13]. Hence, a sufficient degree of flexibility is needed to support dynamic process adaptations in case of unforeseen situations. Finally, *PAIS flexibility* is required to accommodate the need for evolving business processes [23, 22].

In general, a PAIS is aligned in a process-centric way, separating process logic from application code (i.e., the implementation of the application services) and, thus, providing an additional architectural layer [4]. In principle, this separation makes PAISs more flexible compared to data- and function-centric information systems. However, it is not yet sufficient to meet the needs of agile enterprises. In particular, traditional PAIS have focused on the support of predictable and repetitive processes, which can be fully described prior to their execution in terms of formal models [27]. Accordingly, such PAISs require complete specifications (i.e., process models) of the business processes to be supported, which are then used as the schemas for process execution. In practice, however, business processes have become increasingly complex and dynamic, demanding for a more agile approach acknowledging that in dynamic environments process models quickly become outdated and, hence, a closer interweaving of modeling and execution is required. Therefore, PAISs not only need to be able to deal with exceptions [17], change the execution of single business cases on the fly [18], efficiently deal with uncertainty [7], and cope with variability [6, 1], but must also support the evolution of implemented business processes over time [21].

The goal of this paper is to address the flexibility needs emerging in this context and to give insights into technologies addressing them. Emphasis is put on key features enabling process adaptation and evolution. Based on them, PAISs being able to flexibly cope with real-world exceptions, uncertainty and changes can be realized.

2 Traditional Process-Aware Information Systems

A PAIS targets at the operational support of business processes at the IT level. To accomplish this, the business processes need to be mapped to *executable process models*. Thereby, a *business process* comprises a set of one or more connected *activities* that collectively realize a particular *business goal* [15]. A process is linked to an *organizational structure* defining functional roles and organizational relationships. Furthermore, a business process may take place in a specific department, but may also cross departmental borders or even involve different organizations [5]. Examples of business processes include insurance claim processing, order handling, personnel recruitment, product engineering, and patient treatment.

2.1 Business Process Modeling

To provide additional value for the business, any process automation should be preceded by process reengineering and optimiztion efforts [15]; i.e., business processes have to be (re-)designed to meet organizational goals in an economic and efficient manner. Goals pursued may include shortening process cycle times, reducing process costs, increasing customer satisfaction, and decreasing error rates.

To discuss alternative designs with stakeholders and to evaluate the designed processes, process knowledge must be captured in *business process models* [2]. The latter describe business processes at a high level of abstraction, serving as a basis for analysis, simulation and visualization. A business process model comprises the process activities and their attributes (e.g., costs and time) as well as the control and data flow between the activities. Activities may be manual ones without the potential to be automated or system-supported activities requiring human or machine resources for their execution. In general, a distinction has to be made between *business process models* on one hand and their executable counterparts (denoted as *executable process models*) on the other [2]. The latter constitute the key artefacts of a PAIS, realizing the automation of business processes and, in whole or part, the implementation of their models. When interpreting an executable process model, documents, data objects or activities are passed from one actor to another according to pre-defined procedural rules [27]. In the following, we focus on executable process models and their flexible support through PAISs.

2.2 Architectural Principles of a PAIS

A PAIS is a specific type of information system that offers advanced process support services. As opposed to data- or function-centric information systems, PAISs enforce a strict separation of process logic and application code. In particular, process logic is described explicitly in terms of *executable process models* providing the

schema for process execution. Note that turning away from hard-coded process logic towards explicitly specified process models significantly eases (model-driven) PAIS development and maintenance. The core of the process layer of a PAIS, in turn, is built by a process management system. Its buildtime and runtime components offer generic software services for modeling, implementing, executing, and monitoring business processes as well as for enabling user interactions with them (e.g., through worklists). Workow management systems (e.g., ADEPT [4, 19], Staffware [25]) and case handling tools (e.g., FLOWer [25], PHILharmonicFlows [10]) constitute examples of PAISs.

As a basic principle, PAISs foster the splitting of monolithic applications into smaller services, which can then be orchestrated by its *process engine*. Maintainability and traceability are significantly enhanced by this extended architecture. Changes to one layer often can be performed without affecting the other layers. For example, modifying the application service that implements a particular process step (i.e., activity) does usually not imply any change to the process layer as long as interfaces remain stable (i.e., the external observable behavior of the service remains the same). In addition, changing the execution order of activities or adding new activities to the process can, to a large degree, be accomplished without touching the implemenation of any application service.

2.3 Process Enactment Based on Executable Process Models

As already mentioned, the business processes or the process parts to be automated by the PAIS need to be captured in *executable process models*. At buildtime, these models are created based on the elements provided by a process meta model (e.g., BPMN 2.0) using a graphical editor. Basically, an executable process model corresponds to a directed graph that comprises a set of nodes–representing process steps (i.e., activities) or control connectors (e.g, XOR/AND-Split, XOR/AND-Join)–and a set of control edges between them. Control edges specify precedence relations between nodes. Further, the data flow between the activities (i.e., which activities read or write which data elements) needs to be specified and the activities be associated with resources (e.g., user roles). Activities can either be atomic or complex. While an atomic activity is associated with an invokable application service, a complex activity contains a sub-process or, more precisely, a reference to a sub-process model. In turn, this allows for the hierarchical decomposition of process models. Moreover, several executable process models may exist for a particular business process representing the different versions and the evolution of this business process over time. As a benefit of the described model-driven approach, it can be formally checked (e.g., model checking) whether a process model can be properly executed during runtime (e.g., guaranteeing for the absence of deadlocks and ensuring proper data flow). Finally, at runtime the PAIS orchestrates multiple instances of a process model according to the defined logic, also allowing for the integration of application services, users, and other resources.

2.4 Traditional Process Lifecycle Support

Traditional PAISs enable process lifecycle support as depicted in Fig. 1: At build-time, an initial representation of the process to be supported is created either by explicitly modeling the process based on process analysis results or by discovering its model through process mining [26] (1). At runtime, process instances are created from the executable process model (2), each representing a concrete business case. Process instances are executed based on the model they were originally derived from. While fully automated activities are immediately executed when they become enabled, non-automated activities are assigned to the worklists of qualified actors (3). Execution logs record information about the start and completion of activity instances as well as their chronological order (4). The analysis of logs by a process engineer or process intelligence tools allows discovering malfunctions or bottlenecks. In turn, this triggers the evolutionary change of the process model (5).

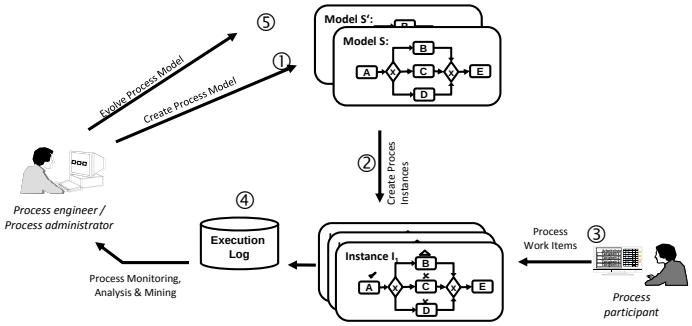

Fig. 1 Process lifecycle support in traditional PAIS

2.5 Key Features of a Process-Aware Information System

In summary, a PAIS

- knows the logic of the supported processes; i.e., processes are explicitly described in terms of executable process models.
- ensures that activities are executed in the specified order or considering the specified constraints (i.e., the PAIS manages the flow of control during runtime).
- controls the flow of data between the activities; i.e., the output data of a particular activity can be consumed as input data by subsequent activities.
- knows the application service to be invoked when an atomic activity is started.

- assigns work items related to human activities to the worklists of authorized users and manages these worklists. Further, it reminds users to complete an activity before reaching its deadline.
- enables end-users to monitor the progress of process instances and to trace their previous execution.
- comprises build- and runtime components that support different stages of the process lifecycle.

3 Enabling Process Flexibility at the Operational Level

The ability to efficiently deal with business process changes has been identied as one of the critical success factors for PAISs [11, 22, 21]. Although PAISs facilitate changes significantly through the separation of concerns, enterprises are reluctant to change PAIS implementations once they are running properly. High complexity and high costs of change are provided as major reasons for not fully leveraging the potential of PAISs. In particular, more flexible PAISs are needed, which enable enterprises to operationalize their processes in a way not causing any mismatch between the digital processes and those running in reality [21]. Moreover, a PAIS must not ”freeze” the implementation of business processes [25], but allow authorized users to flexibly deviate from the pre-specified processes whenever required (e.g., to deal with exceptions) as well as to evolve process implementations over time [17, 23]. Process changes should be enabled at a high level of abstraction [8, 9] without affecting consistency and robustness of the PAIS [18]. Finally, PAISs must allow users to cope with uncertainty by deferring decisions to the runtime if required [21].

Traditional PAISs do not support such advanced scenarios due to their inherent brittleness and inflexibility [25]. What is needed are PAISs that allow both business process implementations and process instances to be continually adapted and reformed to fit the actual needs and constraints of their environment and to fulfill the goals of the involved process participants in the best possible way–we denote such processes as *adaptive*. Traditional PAISs implicitly embrace the ”engineer–use” dichotomy [25] as inherited from traditional approaches to software engineering. This dichotomy is based on the engineering principle that software systems are first *engineered* and then, once deemed fit for purpose, are *used* (i.e., *operated*). Maintenance and evolution tasks are not regarded as part of operation, but rather as interruptions to the ”in use” state, which temporarily return the system to the ”being engineered” state. In scenarios with dynamically emerging or disappearing requirements (e.g. healthcare [16, 13]), this ”engineer–use” strategy is unworkable. The only feasible way to cope with dynamism is to dissolve the fundamental distinction between *engineering* and *use* and to seamlessly merge the entire service and process lifecycle into a single encompassing framework [26]. In turn, this leads to a new class of processes whose *engineering* and *use* is indistinguishable.

4 Adaptive Process-Aware Information Systems

This section reports on *adaptive PAISs*, a next generation technology enabling *adaptive processes* that abandon the ”engineer–use” dichotomy. Adapative PAISs must not be confused with *(self-)adaptive systems* as recognized by the adaptive systems research community [3]. Processes are adaptive in the sense that they are continually evolving and reshaping to fit to the situation at hand, but unlike classical adaptive systems (as understood in adaptive systems' research) they are not expected to do this themselves. On the contrary, the adaptation is performed with the help of the user / engineer. In other words, in *adaptive processes*, human engineers and users are part of the loop, and the use and adaptation of processes are seen as two sides of the same coin. In this sense, *adaptive processes* have more in common with agile software development methods, which focus on encouraging human developers to evolve software in a rapid and effective way. The following sections sketch how adaptive processes and, thus, process flexibility can be realized in PAISs. Note that we do not give detailed insights into formal or technical aspects of adaptive PAISs (see [21, 23, 24, 22, 12]), but want to emphasize the perspectives offered by them, illustrated along the AristaFlow BPM Suite we developed during the last decade.

4.1 The AristaFlow Process Management Technology

During the last decade, we developed the ADEPT2 next generation process management technology [18, 19, 23, 24] to tackle the flexibility challenges discussed in Section 3. ADEPT2 is an adaptive PAIS dissolving the ”engineering–use” dichotomy and increasing ease of use for process implementers, application developers, system administators, and end users. Further, robustness of process implementations and the robust support of dynamic process changes were fundamental project goals. To achieve them, a *correctness-by-construction* principle is applied during process modeling. Furthermore, it is ensured that ad-hoc process instance changes do not introduce any errors or inconsistencies in the following. Due to the high interest of industry in the ADEPT2 technology, it was then transformed into an industrial-strength process management technology called *AristaFlow BPM Suite* [4, 20]. AristaFlow enables robust and flexible PAISs in the large scale. In particular, it ensures error-safe and robust process execution even at the presence of exceptions or dynamic process changes. AristaFlow was applied in a variety of application domains (e.g. healthcare, disaster management, and software engineering).

4.2 Support for Process Adapation and Process Evolution

In general, process adaptations can be accomplished at two levels–the process type and process instance level.

Ad-hoc adaptations at the process instance level. Generally, it is not possible to anticipate all real-world exceptions and to capture their handling in an executable process model at buildtime. AristaFlow, therefore, enables users to situationally adapt single process instances (i.e., specific business cases) during runtime if required; e.g., by inserting, deleting or moving activities [18]. In a medical treatment process, for example, a patient's current medication may have to be discontinued due to an allergic reaction. In general, the effects of ad-hoc changes are instance-specific and must not affect other instances. Providing support for ad-hoc deviations from a pre-specified process model, however, must not shift the responsibility for ensuring PAIS robustness to end-users. Exactly for this reason, AristaFlow provides comprehensive support for the correct, secure and robust handling of runtime exceptions through ad-hoc process instance changes. [21] presents a taxonomy for ad-hoc changes, discusses how the behavior of a process instance can be situationally adapted, and presents adaptation patterns that may be applied for this purpose. Moreover, [21] shows how PAIS robustness can be ensured when dynamically adapting process instances and how end-users can be assisted in defining changes.

Process model evolution and instance migration. Business processes evolve over time due to changes in their legal, technical, or business environment, or as a result of organizational learning [14, 15]. Consequently, PAIS implementations need to be adapted accordingly. We denote this as *process model evolution*, i.e., the evolution of executable process models over time to accommodate changes of real-world processes. In general, process model evolution might require change propagation to already running process instances, particularly if the latter are long-running. For example, let us assume that, due to a new legal requirement, patients have to be informed about potential risks before a surgery takes place. Let us further assume that this change is also relevant for patients for which the treatment has already been started. In such a scenario, stopping all ongoing treatments, aborting them and restarting them is not a viable option. As a large number of treatment processes might be running at the same time, applying this change manually to all ongoing treatment processes is also not feasible. AristaFlow, therefore, provides efficient support to add this step to all patient treatments for which this is still feasible (e.g., if the surgery has not yet started). For this purpose, it offers techniques for dealing with already running process instances and their on-the-fly migration to the changed process model, without violating any correctness and soundness properties. In this context, well-known process adaptation patterns may be applied, which provide precise pre- and post-conditions for ensuring syntactical correctness and behavioral soundness of a process model, i.e., a correctness-by-construction principle is applied [18, 4]. Deficiencies that cannot be prohibited by this approach (e.g., correctness of the data flow schema) are checked on-the-fly and are continuously reported to the user.

In general, process model evolution and instance-specific ad-hoc changes have to be handled in combination with each other [23, 24, 26]. Moreover, AristaFlow provides built-in-flexibility allowing process engineers to leave parts of the process model unspecified at buildtime and to add the missing information during runtime. Especially, this approach is useful in case of uncertainty as it allows deferring decisions from build- to runtime.

4.3 Advanced Process Lifecycle Support in Adaptive PAIS

The described ability of AristaFlow for enabling ad-hoc changes in a controlled, correct and secure way as well as for the controlled evolution of process models (including process instance migrations) leads to a revised process lifecycle [26] (cf. Fig. 2): At buildtime, an initial representation of a business process is created, either by modeling the process or by discovering its model through process mining (1). New process instances can be derived at runtime from this executable process model (2). Instances are executed according to the original process model they were derived from, and activities are assigned to process participants to perform the respective activities (3). However, when unanticipated exceptional situations occur during runtime, process participants may deviate from the pre-specified model by applying ad-hoc changes (4). While execution logs record information about activities (3), process changes are recorded in change logs and may be semantically represented as cases (4). The latter enables the reuse of ad-hoc changes in similar situations [26]. The analysis of these logs by process engineers or process intelligence tools allows for the discovery of malfunctions or bottlenecks, which often leads to an evolution of the process model (6). The latter is supported through versioning as well as the ability of dynamically migrating already running process instances.

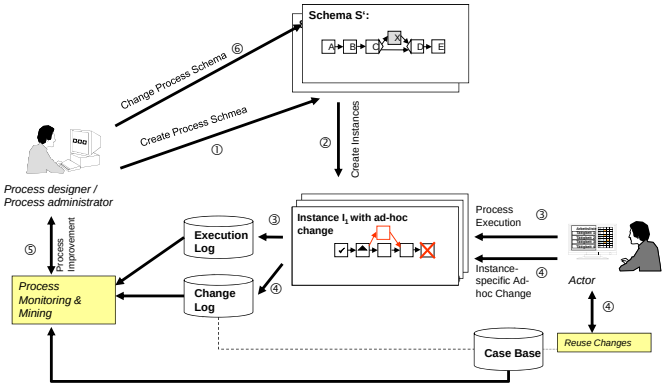

Fig. 2 Process lifecycle support in adaptive PAIS

4.4 Making Process Implementations Flexible and Robust

We now focus on a fundamental pillar of any robust process implementation, i.e., *error handling*. In particular, we show how the presented process adaptation features can be utilized to make business process implementations flexible and robust.

4.4.1 Error Prevention

AristaFlow targets at *error prevention*, which is achieved by applying a *correctness-by-construction* principle during process modeling and service composition as well as by guaranteeing correctness and robustness in connection with dynamic process changes. The latter means that none of the PAIS correctness properties ensured by respective checks at buildtime may be violated due to a dynamic process change. This was probably the most inuential challenge for our research. It also had signicant impact on the development of the AristaFlow BPM Suite. In particular, we try to detect as many errors as possible at buildtime (e.g., flaws in the data flow or deadlocks) to exclude their occurence during runtime. As discussed, however, errors cannot be always prevented. Therefore, another important aspect of PAIS robustness concerns its exception handling features. We will show that the AristaFlow BPM Suite provides an easy, but yet powerful way to handle exceptions during runtime. In this context, the ability to support ad-hoc process changes is very useful. By utilizing such dynamic changes, it becomes possible to even cope with severe process failures and to continue and complete respective process instances.

We will use an example to demonstrate how errors can be handled in the AristaFlow BPM Suite. Consider Fig. 3, 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 the given scenario, he will request an offer from Amazon using a web service and another offer from a another vendor using e-mail. After receiving 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 a corresponding offer to his customer.

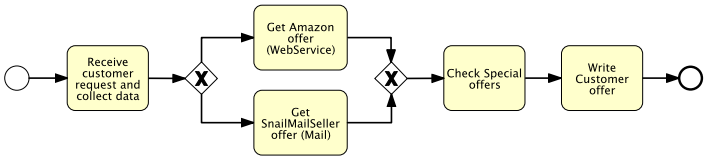

Fig. 3 Scenario: A simple process calling a web service (in BPMN notation)

The scenario contains several sources of potential errors. While some of them can be addressed at buildtime, others cannot. For example, assume that the process implementer does not foresee a way to enter the offer from *SnailMailSeller* into the system. Then, the final activity might fail or produce an invalid output as its input parameters are not set properly. Another source of error might be the Amazon web service, e.g. it might not be available when making the request and, therefore, activity *Get Amazon offer* might fail at runtime. Such errors can be foreseen and, hence, be considered at buildtime. However, unexpected errors might occur as well; e.g., *Check Special offers* might fail due to troubles with the Internet connection.

The following requirements for error-safe and robust process execution exist: On one hand, errors should be avoided at buildtime, on the other, PAIS should enable users to effectively deal with both expected and unexpected errors during runtime.

4.4.2 Error and Exception Handling in the AristaFlow BPM Suite

We consider the above example from the perspectives of the *process implementer* (i.e., the process engineer), the *system* (i.e., the PAIS), the *end user* (i.e., the process actor), and the *system supervisor* (i.e., the PAIS administrator). We discuss how each of these parties can contribute to the handling of errors.

Process implementer perspective

Fig. 4 shows a part of the process from Fig. 3, as it can be modeled using the *AristaFlow Process Template Editor*. For process implementation, the idea of process composition in a *plug & play* style is pursued and supported by comprehensive correctness checks. The latter aims to exclude runtime errors during process execution. As prerequisite, for example, the data flow dependencies among application services have to be made explicit to the PAIS.

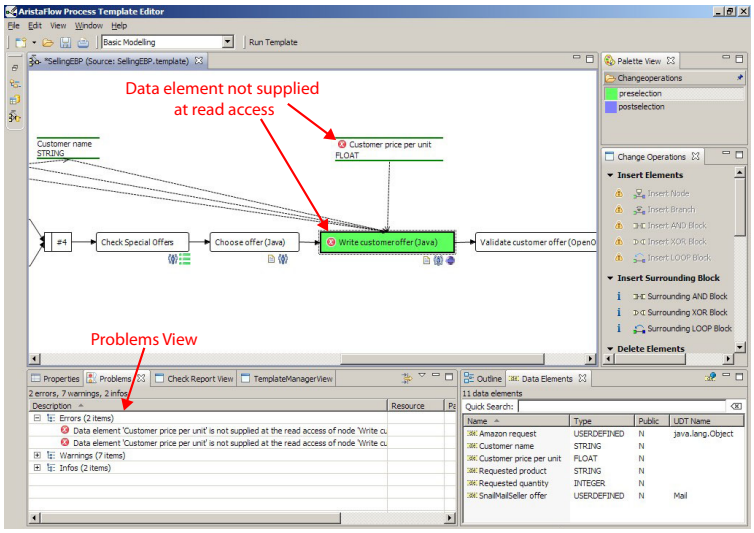

Fig. 4 AristaFlow Process Template Editor

AristaFlow provides an intuitive graphical editor and composition tool to process implementers (cf. Fig. 4). Further, it applies a *correctness-by-construction* principle

by providing at any time only those change operations to the user, which allow transforming a sound process model into another one; i.e., change operations are enabled or disabled depending on which region in the process graph is marked for applying an operation. Deficiencies not prohibited by this approach (e.g., regarding data flow) are checked on-the-fly and are reported continuously in the problem window of the *Process Template Editor*. An example is depicted in Fig. 4, where AristaFlow detects that data element *Customer price per unit* is read by activity *Write Customer offer*, but not written by any preceding activity.

In general, one should not require detailed knowledge from process implementers about the internals of the application services they may assign to the activities of an executable process model. However, this should not be achieved by undermining the *correctness-by construction* principle. In AristaFlow, all kinds of executables (e.g., web services, SQL procedures, Java Apps), which may be assigned to process activities, first have to be registered in the *Activity Repository* as activity templates. An activity template, in turn, provides all information to the *Process Template Editor*, e.g., information about mandatory and optional input/output parameters of activities 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.

As a major advantage of this approach, common errors (e.g., missing data bindings) can be already detected at buildtime. Consequently, the time needed for testing and debugging process implementations can be signicantly reduced; i.e., AristaFlow guarantees that executable process models without any detected deficiencies are sound and complete with respect to the activity templates used.

System perspective

The described approach ensures that, in principle, the process model is executable by the PAIS in an error-safe way. As always, this might not hold in practice. Again, consider the scenario from Fig. 3. The web service referred by activity *Get Amazon offer* (i.e., the service implementing this activity) might not be available when the process is executed, leading to an exception during runtime. Note that such errors neither can be detected in advance nor be completely prevented by the PAIS.

Failures of the Amazon web service might be anticipated by the process implementer. Thus, he may assign specific error handling procedures to the respective activity. Following a strict process paradigm, AristaFlow itself uses processes to coordinate exeception handling, i.e., a *reflective* approach is taken in which error handling is accomplished by a specific process executed in AristaFlow. A simple error handling process is depicted in Fig. 5. Depending on whether the failure of the activity was triggered by the user (e.g. through an *abort* button) either the system supervisor is notified accordingly or the process silently terminates. Generally, error handling processes can become arbitrarily complex and long-running. Note that AristaFlow treats error handling processes the same way as any other process. Thus, they may refer to any activity registered in the repository. Note that this allows for

error handling at a higher semantical level, involving users whenever required. If an activity fails, the respective error handling process is initiated and equipped with all the data necessary to identify and handle the error, e.g. the ID of the failed activity instance, the actors responsible for the activity, or the cause of the error (cf. Fig. 5).

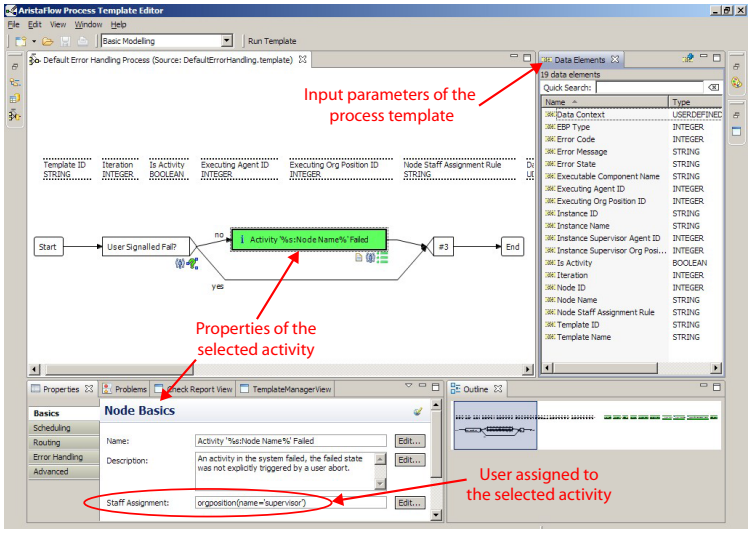

Fig. 5 A simple error handling process

After creating an error handling process and deploying it 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. Note that it is also possible to assign an error handling process to a complete process model instead of assigning it to a specific activity. Then, this general error handling process will be used if no other error handling process is associated with a failed activity. If no error handling process is assigned to the activity and process, in turn, a system default error handling process will be used instead.

As a considerable advantage of using processes for error handling, standard process modeling tools and techniques can be used for designing error handling strategies. Therefore, process implementers need not learn any new concept to provide sophisticated error handing procedures. As another important advantage, error handling at a higher semantical level becomes possible. For example, one may also realize more complex error handling strategies like compensation or apply ad-hoc changes to replace parts of the failed process.

End user perspective

The error handling process from Fig. 5 might not be always appropriate as it increases the workload of the system supervisor. Most standard errors can be handled in a (semi-)automatic way by the actor executing the activity. Upon failure of the activity, the actor responsible for its execution could be provided with a set of possible error handling strategies among which he may choose. An example of such a more complex error handling process is shown in Fig. 6. Here, the user may choose between a variety of ways to handle the error, e.g., retrying the failed activity, aborting the entire process instance, or applying pre-specified ad-hoc changes to fix or compensate the error. Moreover, the error handling strategies suggested in a particular context may depend on the background of the respective user, i.e., on his knowledge, organizational position, and various other factors. Depending on the selected user, the respective strategy is chosen and applied to handle the error.

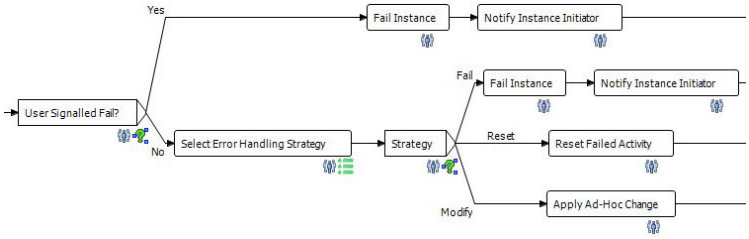

Fig. 6 A more complex error handling process involving the user

The described semi-automatic approach provides several advantages. As for each activity a predefined set of strategies can be offered to users, they need not have deep insights into the process of properly handling errors. This allows reducing waiting times for failed activity instances as users themselves can handle errors immediately without waiting for a busy helpdesk. In turn, this relieves the helpdesk from the tedious task of dealing with simple process errors.

System supervisor perspective

Certain errors cannot be handled by the user. For example, errors 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 may use the *AristaFlow Process Monitor* as shown in Fig. 7 to take a look at this process instance, to analyze its execution log, and to decide for an appropriate error handling strategy. Additionally, the system supervisor may use the *AristaFlow Process Monitor* to keep track of failed instances; e.g., he may intervene if a web service becomes unavailable permanently.

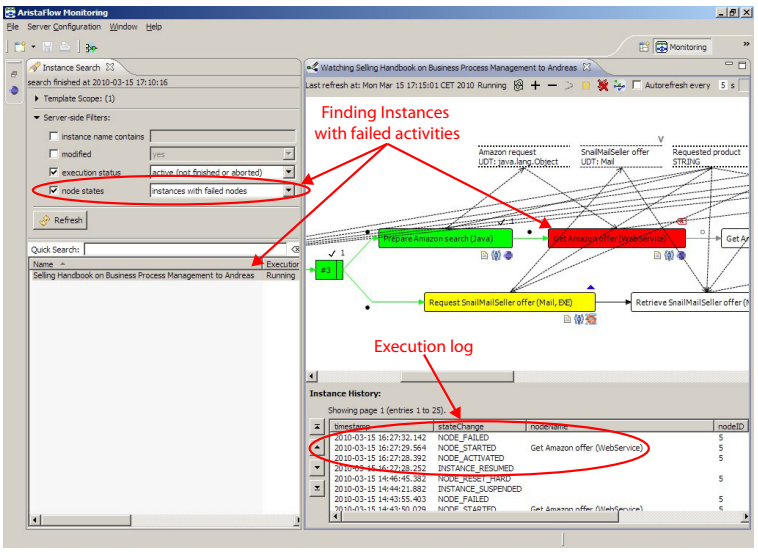

Fig. 7 AristaFlow Process Monitor: Monitoring Perspective

Reconsider the bookseller scenario from Fig. 3. 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 administrator detects that the process instance is in trouble and uses the *AristaFlow Process Monitor* to take a look at it (cf. Fig. 7). 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 such 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 result of this and the subsequent activity is only needed when executing the *Choose oer* activity. Therefore, he dynamically moves these two activities after activity *Check Special Oers*; i.e., the user may continue working on this process instance before the PAIS tries to reconnect to Amazon (cf. Fig. 8).

To realize the described change, he can switch to the *Instance Change Perspective* of the *AristaFlow Process Monitor*, which provides the same set of change operations as the *Process Template Editor*. In fact, it is the *Process Template Editor* being aware that a process instance has been loaded and, therefore, instance-related state information is additionally taken into account when enabling/disabling change operations and applying correctness checks (e.g., the application of changes to already passed regions of the respective process model would be prohibited). The system administrator would now move the two activities to their new position by applying the respective change operation. The resulting instance is depicted in Fig. 8. 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

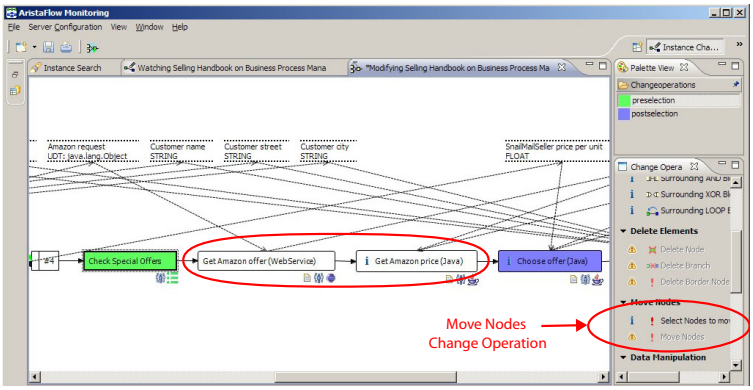

Fig. 8 AristaFlow Process Monitor: Instance Chance Perspective

the system administrator to delete the activities being in trouble and to replace them with a form-based activity allowing him to enter the price manually.

5 Conclusions

Adaptive processes fundamentally change the way in which human stakeholders interact and collaborate as they dissolve the distinction between process engineers and end-users. To date, business process support technologies have focused on enhancing and automating the way in which process users collaborate and interact, but have not significantly changed the way in which the processes themselves are engineered (i.e. defined and maintained). It has been assumed that this is done by IT specialists in a distinct engineering phase with little or no connection to the execution of the processes or the normal operation of the enterprise. However, with adaptive processes this distinction will blur (if not entirely disappear) and process engineers will become process users and vice versa. Stated differently, process engineering will be also regarded as a normal adaptive process involving the collaboration of multiple stakeholders.

References

1. Ayora, C., Torres, V., Weber, B., Reichert, M., Pelechano, V.: VIVACE: A framework for the systematic evaluation of variability support in process-aware information systems. Inf Softw Techn, Elsevier, 57, 248-276 (2015)
2. Buchwald, S. et al: Bridging the gap between business process models and service composition specifications. In: Service Life Cycle Tools and Technologies, 124-153 (2011)

3. Cheng, B., et al.: Software engineering for self-adaptive systems: A research roadmap. In: Software engineering for self-adaptive systems, Springer, 1-26 (2009)
4. Dadam, P., Reichert, M.: The ADEPT project: a decade of research and development for robust and flexible process support. Comp Sci Res Develop, Springer, 23(2), 81-97 (2009)
5. Fdhila, W., Indiono, C., Rinderle-Ma, S., Reichert, M.: Dealing with change in process choreographies: design and implementation of propagation algorithms. Inf. Sys., 49, 1-24 (2015)
6. Hallerbach, A., Bauer, T., Reichert, M.: Capturing variability in business process models: the Provop approach. J Softw Maint Evol, Wiley, 22(6-7), 519-546 (2010)
7. Haisjackl, C., Barba, I., Zugal, S., Soffer, P., Hadar, I., Reichert, M., Pinggera, J., Weber, B.: Understanding Declare models: strategies, pitfalls, empirical results. Softw Sys Modeling, Springer, 15(2), 325-352 (2016)
8. Kolb, J., Kammerer, K., Reichert, M.: Updatable process views for user-centered adaption of large process models. In: Proc. ICSOC'12, Springer, LNCS 7636, 484-498 (2012)
9. Kolb, J., Reichert, M.: A flexible approach for abstracting and personalizing large business process models. Appl. Comp. Review, ACM SIGAPP, 13(1), 6-17 (2013)
10. Künzle, V., Reichert, M.: PHILharmonicFlows: towards a framework for object-aware process management. J Softw Maint Evol, Wiley, 23(4), 205-244 (2011)
11. Künzle, V., Weber, B., Reichert, M.: Object-aware business processes: fundamental requirements and their support in existing approaches. J Inf Sys Modeling Design, 2(2), 19-46 (2011)
12. Lanz, A., Weber, B., Reichert, M.: Time patterns for process-aware information systems. Requirments Engineering, Springer, 19(2), 113-141 (2014)
13. Lenz, R., Reichert, M.: IT support for healthcare processes - premises, challenges, perspectives. Data Knowledge Engineering, Elsevier, 61(1), 39-58 (2007)
14. Li, C., Reichert, M., Wombacher, A.: Mining business process variants - challenges, scenarios, algorithms. Data Knowledge Engineering, Elsevier, 70(5), 409-434 (2011)
15. Lohrmann, M., Reichert, M.: Effective application of process improvement patterns to business processes. Softw Sys Modeling, Springer, 15(2), 353-375 (2016)
16. Reichert, M.: What BPM technology can do for healthcare process support. In: Proc. AIME'11, Springer, LNAI 6747, 2-13 (2011)
17. Reichert, M., Dadam, P., Bauer, T.: Dealing with forward and backward jumps in workflow management systems. Softw Syst Modeling, 2(1): 37-58 (2003)
18. Reichert, M., Dadam, P.: ADEPTflex - supporting dynamic changes of workflows without losing control. J Intellig Inf Sys, 10(2), 93-129 (1998)
19. Reichert, M., Dadam, P.: Enabling adaptive process-aware information systems with ADEPT2. In: Handbook of Research on Business Process Modeling, IGI, 173-203 (2009)
20. Reichert, M., et al: Enabling Poka-Yoke workflows with the AristaFlow BPM Suite. In: Proc. BPM'09 Demonstration Track, CEUR Workshop Proceedings 489 (2009)
21. Reichert, M., Weber, B.: Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies. Springer, Heidelberg (2012)
22. Reichert, M., Rinderle-Ma, S., Dadam, P.: Flexibility in process-aware information systems. ToPNoC, Springer, LNCS 5460, 115-135, (2009)
23. Reichert, M., Rinderle, S., Dadam, P.: On the common support of workflow type and instance changes under correctness constraints. In: Proc. CooplS '03, Springer, LNCS 2888, 407-425 (2003)
24. Rinderle, S., Reichert, M., Dadam, P.: Disjoint and overlapping process changes: challenges, solutions, applications. In: Proc. CooplS'04, Springer, LNCS 3290, 101-121 (2004)
25. Weber, B., Mutschler, B., Reichert, M.: Investigating the effort of using business process management technology: results from a controlled experiment. Sci. Comp. Prog., 75(5), 292-310 (2010)
26. Weber, B., Reichert, M., Wild, W., Rinderle-Ma, S.: Providing integrated life cycle support in process-aware information systems. J Coop Inf Sys, World Sci Publ, 18(1), 115-165 (2009)
27. Weske, M.: Business Process Management - Concepts, Languages, Architectures, 2nd Edition, Springer (2012)