Workflow Management

Chun Ouyang, Michael Adams, Moe Thandar Wynn, and Arthur H.M. ter Hofstede

Abstract Workflow management has its origin in the office automation systems of the seventies, but it is not until fairly recently that conceptual and technological breakthroughs have led to its widespread adoption. In fact, nowadays, process-awareness has become an accepted and integral part of various types of systems. Through the use of process-aware information systems, workflows can be specified and enacted, thus providing automated support for business processes. A workflow explicitly represents control-flow dependencies between the various tasks of the business process, the information that is required and that can be produced by them, and the link between these tasks and the resources, be they human or not, which can execute them. In this way, processes can be performed more efficiently and effectively, compliance with respect to standard procedures and practices can be monitored more closely, and rapid change in response to evolving market conditions can be achieved more easily. This chapter provides an overview of the field of workflow management.

1 Introduction

Workflow management is concerned with providing automated support for business processes. Typically, a workflow involves both people and software applications. Work is assigned to participants based on explicit resource allocation directives, which may link into an organizational model, and the timing is driven by an explicit representation of the temporal order of the various activities of the business process.

C. Ouyang (✉)
Business Process Management Group, Faculty of Science and Technology, Queensland University of Technology, Brisbane, Australia
e-mail: c.ouyang@qut.edu.au

J. vom Brocke and M. Rosemann (eds.), *Handbook on Business Process Management 1*, International Handbooks on Information Systems,
DOI 10.1007/978-3-642-00416-2_18,

Apart from the obvious fact that there is potential for savings in terms of time and money, there are other benefits in deploying workflow applications. By having explicit representations of these resource and control-flow dependencies, it can be claimed that changing workflows is easier and hence a business that has automated its processes by means of a workflow management system may be more responsive to changes in its environment, such as changing legislation or evolving market conditions. As workflow management systems log events that pertain to business processes (e.g., the fact that a certain resource has completed a certain task at a certain point in time), process logs may be used to demonstrate that a business complies with best practices or with existing legislation. Log files provide a valuable starting point for process analysis and for subsequent process improvement. The area of *process mining* (van der Aalst et al. 2004b) is concerned with process-related information that can be derived from log files.

The Workflow Management Coalition[1] has defined what the components of a workflow environment are and what interfaces these components should have to support interaction with each other and with external components (Fischer 2005). In a workflow management environment, there is typically a component that supports the specification of workflows and another that supports the execution of these workflows. There are also, usually, components that can deal with external applications or other workflow engines or that provide support for administration and monitoring.

A workflow can be examined from a number of perspectives (van der Aalst et al. 2003; Jablonski and Bussler 1996). The temporal order of the various tasks in a workflow can be referred to as the *control-flow perspective*. The way data is defined and passed between workflow elements and/or the external environment is captured in the *data perspective*. The *resource perspective* is concerned with controlling the way resources become involved in the execution of tasks. Naturally, these perspectives are related, e.g., a missing data item may hold up the execution of a certain task or the resource selected for the execution of a certain task may be determined on the basis of the number of times they have performed this task in the past. Understanding the role of these perspectives is vital to understand what workflow management is about.

In this chapter, we aim to provide the reader with an overview of concepts and technology that underlie modern workflow management. We will start by exploring the conceptual foundations of workflow management, which will inform the subsequent discussion of a number of approaches to workflow specification. More advanced topics follow, dealing with change and unexpected exceptions, simulation, verification, and configuration, after which we discuss an existing workflow management system that can be seen as a reference implementation for some state-of-the-art concepts. The aim of presenting this system is to reinforce the understanding of concepts discussed. The chapter ends with a case study in the domain of screen business, followed by a brief overall conclusion.

[1] http://www.wfmc.org

1.1 An Introductory Example

A workflow, sometimes used as a synonym for "a business process," comprises a series of tasks (activities) through which work is routed. Workflow management systems are a class of software that supports business processes by taking on their information logistics, i.e., they ensure that the right information reaches the right person at the right time (van der Aalst and van Hee 2002). The information logistics of business processes can be captured by a workflow or process modeling language. Different workflow management systems may be implemented supporting the use of different languages.

Consider an example of a process that models a credit card application. The process starts when an applicant submits a credit card application (Task 1). Upon receiving the application, a clerk examines if the requested loan amount is large (e.g., greater than $5000) or small (Task 2) and then performs different eligibility checks accordingly (Task 3 for large loan and Task 4 for small loan). Let us stop here for the moment (we will continue describing the process in the languages section). It can be observed that there are dependencies between the above tasks. Task 1 is (sequentially) followed by Task 2, and after Task 2, an exclusive choice is made, determining whether to perform Task 3 or Task 4. A workflow language can be used to capture these in a precise manner. However, many workflow languages exist due to lack of consensus. For example, as Fig. 1 illustrates, the flow comprising the above tasks in a credit card application process can be captured using five mainstream workflow or process modeling languages: BPMN (Business Process Modeling Notation) (Fig. 1a), EPC (Event-driven Process Chain) (Fig. 1b), BPEL (Business Process Execution Language for Web Services) (Fig. 1c), Petri nets (Fig. 1d), and YAWL (Yet Another Workflow Language) (Fig. 1e). We shall describe these in more detail in the languages section. For the moment, it is sufficient to observe that in Fig. 1, each of these languages models the same exclusive-choice behavior (i.e., XOR-split) in a different way.

The exclusive choice is just one of many recurring modules that may exist in business processes. So, is there a way to identify these modules in a language- and system-independent manner? In the next section, we answer this question by introducing the concept of workflow patterns.

2 Workflow Patterns

Workflow patterns are a specialized form of *design patterns* defined in the area of software engineering. They refer specifically to recurrent problems and proven solutions related to the development of process-oriented applications in both a language- and technology-independent manner. The Workflow Patterns Initiative[2]

[2] http://www.workflowpatterns.com

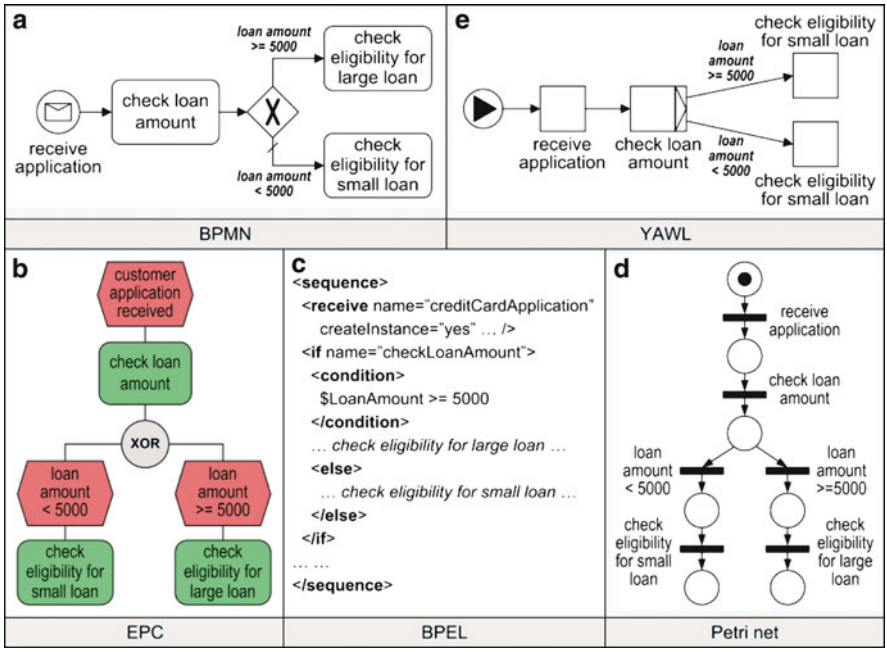

Fig. 1 Modelling the first four tasks in an example of a credit card application process using each of five mainstream workflow or process modeling languages

was established in the late 1990s with the aim of delineating the fundamental requirements that arise during business process modeling on a recurring basis and describing them in an imperative way.

Originally, a set of twenty patterns was identified describing the control-flow perspective of business processes (van der Aalst et al. 2003). These patterns capture structural characteristics of a business process and the manner in which the thread of execution flows through the process model. Since their release, they have been widely used by practitioners, vendors, academics alike in the selection, design and development of workflow systems, and standards. For example, they were used to evaluate 15 commercial workflow systems including such as IBM's WebSphere, Staffware Process Suite, and the case handling system FLOWer. Established process modeling languages such as Petri nets, EPCs, and UML Activity Diagrams (both versions 1.4 and 2.0) were also subjected to a pattern-based evaluation. In addition, vendors and organizations performed analysis of their tools or standards based on workflow patterns. Examples include White's report (White 2004) showing how BPMN supports the original control-flow patterns and TIBCO's report on how Staffware realizes these patterns, to name a few.[3]

[3]See an up-to-date list of vendors' evaluations of tools and standards in terms of the original twenty control-flow patterns at http://www.workflowpatterns.com/vendors.

Later, a detailed review of the original 20 patterns led to the identification of 23 new patterns (Russell et al. 2006b). In total, the 43 control-flow patterns can be classified into eight categories: basic control-flow, advanced branching and synchronization, multiple instances, state-based situations, iteration, external triggering, cancelation, and termination. For example, one of the advanced synchronization patterns is called the General Synchronizing Merge (or OR-join). The OR-join synchronizes only if necessary, i.e., it will synchronize only the active incoming branches and it is certain that the remaining incoming branches, which have not been enabled, will not be enabled at any future time. In general, this synchronization decision cannot be made *locally*. It requires awareness of both the current state and possible future states for the current process instance. Another example is the Deferred Choice, one of the state-based patterns. It captures the scenario when the choice among a set of alternative conditional branches is based on interaction with the operating environment. The decision is delayed until the first task in one of these branches is initiated, i.e., there is no explicit choice but rather a race between different branches.

In addition to the control-flow patterns, workflow patterns have also been extended to cover the data and resource perspectives. There are 40 data patterns (Russell et al. 2005b) capturing a series of data characteristics that occur repeatedly in business processes. These cover data visibility (e.g., scoping of data variables), data interactions within a business process (internal) or between the process and its operating environment (external), data transfer between one process component and another, and data-based routing that describes how data elements can interact with other perspective (particularly the control-flow perspective) and influence the overall operation of a process instance.

For the resource perspective, 43 patterns (Russell et al. 2005a) have been identified, capturing the various ways in which resources are represented and utilized in business processes. Based on the lifecycle of a work item (which include resourcing states such as *offered*, *allocated*, *started,* and *completed*), the resource patterns can be classified into seven categories: creation patterns for design-time work allocation, push patterns for system distributing work items to resources, pull patterns for resources identifying to executing work items, detour patterns for work item rerouting, auto-start patterns for automated commencement of work items based on criteria, visibility patterns for configuration of the visibility of work items for certain participants, and multiple resource patterns for work allocations involving multiple participants or resources. For example, one of the detour patterns is called the delegation pattern. It captures the scenario where a resource allocates an unstarted work item that was previously allocated to it to another resource. This provides a resource with a means of rerouting work items that it is unable to execute (e.g., the resource is going to be unavailable).

Finally, there are also patterns for exception handling, which deals with the various causes of exceptions and the various actions that need to be taken as a result of exceptions occurring. This will be described later in the chapter.

3 Languages

Workflow languages are used to design workflow models in order to capture processes at a level of detail that is sufficient to enable their execution (van der Aalst and van Hee 2002; Weske 2007). Examples include: dedicated workflow specification languages such as XPDL and YAWL; executable process definition languages based on Web services such as BPEL and XLANG; and workflow products such as Staffware and IBM's Websphere. It is also possible to use languages designed for business process modeling, such as BPMN and EPC, to specify workflows. However, for process execution, these models need to be transformed to models specified in an executable language such as BPEL or YAWL.

In this section, we firstly introduce BPMN and BPEL, which are considered as two mainstream languages for capturing business processes from a practitioner's point of view. We then move onto YAWL, which is developed in the academic domain and supports most workflow patterns identified so far. YAWL can be seen as state of the art in the domain of workflow languages. It is therefore used to illustrate the main concepts in the field of workflow management in this chapter.

3.1 BPMN and BPEL

BPMN is a business processing modeling notation intended to facilitate communication between domain analysts and to support decision making based on techniques such as cost analysis, scenario analysis, and simulation. Process models specified in BPMN are therefore not meant to be directly executable. On the other hand, BPEL is intended to support the definition of a class of business processes for Web service interactions. The logic of the interactions is described as a composition of communication actions that are interrelated by control-flow dependencies expressed through constructs corresponding to parallel, sequential, and conditional execution, event, and exception handling. BPEL allows for the specification of executable business processes, and therefore can be used to support the execution of BPMN models.

The use of BPMN (for process modeling) in conjunction with BPEL (for process execution) is a typical example of the approach where two different languages are used, respectively, for the modeling and execution stages and thus a transformation between these languages is required. There are obvious drawbacks to this separation of modeling and execution, especially when both languages are based on different paradigms or when the modeling language contains potentially complex concepts and little consideration was given to their precise meaning. For example, BPMN is graph-oriented, which means that a model captured in BPMN can have an arbitrary topology, while BPEL is block-structured; thus, if a segment of a BPEL model starts with a branching construct, it ends with the corresponding synchronization construct. A mapping from BPMN to BPEL, such as the one proposed in

Ouyang et al. (2009), needs to handle the above mismatches properly and may still result in BPEL code that is hard to understand.

3.2 YAWL and Its Formal Foundation

As mentioned in the previous section, the original 20 control-flow patterns were used to evaluate various workflow and process modeling languages, standards, and workflow products. The evaluation results showed that Petri nets have at least three distinct advantages for being used as a workflow language: formal semantics, state-based instead of (just) event-based, and abundance of analysis techniques (van der Aalst 2000). They are quite expressive compared to many process languages, e.g., they offer direct support to all state-based patterns. Nevertheless, there are serious limitations in Petri nets (as in other languages) when it comes to capturing three categories of patterns: (1) patterns involving multiple instances, (2) advanced synchronization patterns (e.g. OR-join), and (3) cancelation patterns. For example, patterns involving multiple instances capture scenarios where within the context of a single workflow instance (i.e., case), part of the process (e.g., a task or a sub-process) need to be instantiated multiple times, e.g., within the context of an academic paper review, multiple reviewers need to review the paper, and these review results will then be used to determine the final result. The number of multiple instantiations may be known a priori at design-time/runtime, or not be known at all until the process proceeds to the next part (at runtime). In high-level Petri nets, it is possible to use advanced constructs to capture multiple instances of a task or a subprocess. However, there is no specific support for *patterns involving multiple instances*, and the burden of keeping track of splitting and joining the various multiple instances is borne by the designer.

The observation of the limitations in Petri nets for capturing certain workflow patterns triggered the development of a new language – YAWL. YAWL took Petri nets as a starting point and introduced mechanisms that provide direct support for the control-flow patterns especially the above three categories of patterns.

3.2.1 Petri Nets

A Petri net (Murata 1989) is a directed graph composed of two types of nodes: *places* and *transitions*. Usually, places are represented as circles and transitions as rectangles. Petri nets are bipartite graphs, meaning that an arc in the net may connect a place to a transition or vice versa, but no arc may connect two nodes of the same type. A transition can have a number of immediately preceding places (called it *input places*) and a number of immediately succeeding places (called its *output places*).

Places may contain zero or more *tokens*, which model the thing(s) that flow through the system. The state, often referred to as *marking*, is the distribution of

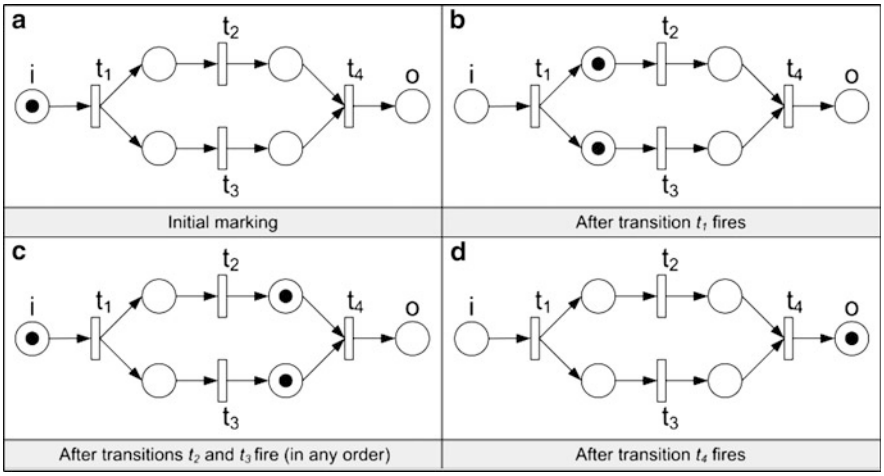

Fig. 2 A sample Petri-net in four different markings

tokens over places. For example, Fig. 2a depicts an initial marking of a Petri net where there is one token in the leftmost place *i* and no token in any other place. The state of a Petri net changes when one of its transitions fires. A transition may fire if there is at least one token in each of its input places. In this case, we say that the transition is *enabled*. For example, in Fig. 2a, the transition labeled t_1 is enabled since it has only one input place and this input place has one token. When a transition fires, it removes one token from each of its input places and adds one token to each of its output places. For example, Fig. 2b depicts the state obtained when transition t_1 fires starting from the initial marking in Fig. 2a. The token in place *i* has been removed, and a token has been added to each of the output places of transition t_1. In a given marking, there may be multiple enabled transitions simultaneously. In this situation, any of these transitions may fire at any time. For example, in Fig. 2b, two transitions t_2 and t_3 are enabled, and any of them may fire in the next execution step. After both t_2 and t_3 fire, transition t_4 is enabled (Fig. 2c), and after t_4 fires, the net reaches a final marking where only the rightmost place *o* holds a token and none of the transitions are enabled.

It can be observed that in the Petri net shown in Fig. 2, transition t_1 behaves like an AND-split, transition t_4 behaves like an AND-join, and transitions t_2 and t_3 capture concurrent executions of two parallel branches. In comparison to this, Fig. 3 depicts two examples of Petri nets modeling executions of conditional branches. In each net, the output place of transition t_1 is the input place of two transitions. When there is a token in this place, the two transitions sharing the place are both enabled, but only one of them may fire, i.e., firing of one of the two transitions will consume the token, thus disabling the other transition. In Fig. 3, the difference between the two Petri nets is with regard to how the choice is made among the conditional branches. In Fig. 3a, the choice can be made (explicitly) by the system upon evaluating the condition *c*. If *c* evaluates to true, transition *c* will fire; otherwise,

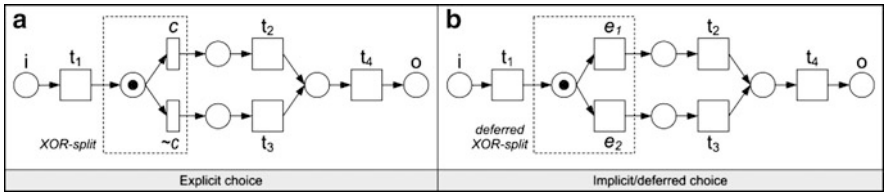

Fig. 3 The sample Petri-nets capturing two types of choices between conditional branches

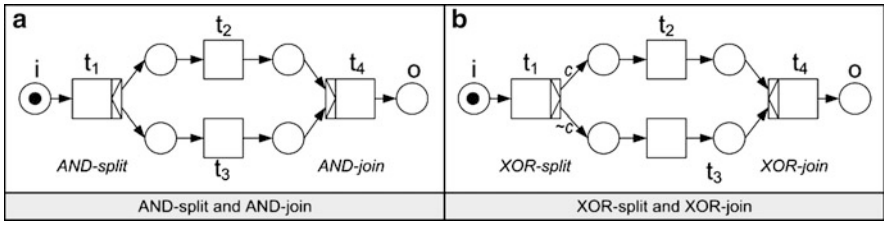

Fig. 4 WF-net notations for AND/XOR splits and joins

transition $\sim c$ will fire. In Fig. 3b, the choice is deferred until one of the events e_1 and e_2 occurs, e.g., e_1 may indicate the arrival of an external message, and e_2 may signal a timeout. Triggers of such events come from the environment.

3.2.2 Workflow Nets

Workflow nets (WF-nets) (van der Aalst 1997) are a subset of Petri nets used to model workflows. A WF-net satisfies the following requirements: there is a unique input place (i) and a unique output place (o), and every other place and transition are on a directed path from place i to place o. In other words, WF-nets have a distinct start place and a distinct end place. For example, the Petri nets in Figs. 2 and 3 are all WF-nets. Intuitively, a WF-net models the execution of one instance of a business process. The initial marking of a WF-net contains a single token in the start place, and in principle, at least one token should reach the end place.

In a WF-net, special notations are introduced to illustrate constructs such as AND-split, AND-join, XOR-split, and XOR-join due to their frequent occurrences in modeling workflows. Figure 4 depicts these notations using the WF-nets shown in the previous figures. In Fig. 4a, the WF-net in Fig. 1 is redrawn (without affecting the behavioral semantics of the net) by replacing transition t_1 with an AND-split and t_4 with an AND-join. In Fig. 4b, the WF-net in Fig. 3a is redrawn using XOR-split and XOR-join. The XOR-split (t_1) captures the fact that after t_1 occurs, a token must be produced for one of its output places (based on the evaluation result of condition c). The XOR-join (t_4) is enabled if one of its input places contains a token. Alternatively, an XOR-join can also be modeled by a place, e.g., the input place of transition t_4 in Fig. 3.

3.2.3 YAWL

YAWL (van der Aalst and ter Hofstede 2005) extends the class of WF-nets with three categories of patterns: multiple-instance, OR-join, and cancelation patterns. In contrast to Petri nets and WF-nets, the syntax of YAWL allows tasks to be directly connected, which helps compress the visual representation of a YAWL model. Figure 5 shows the modeling elements of YAWL. A process definition in YAWL consists of *tasks* (i.e., transition-like objects) and *conditions* (i.e., place-like objects). Each process definition starts with a unique *input condition* and a unique *output condition*.

A workflow specification in YAWL is a set of workflow nets which forms a directed rooted graph. There are *atomic tasks* and *composite tasks*. Both types of task can also be *multiple instance* tasks at the same time and thus have multiple concurrent instances at runtime. Each composite task refers to a net that contains its expansion. Atomic tasks correspond to atomic actions, i.e., actions that are either performed by a user or by a software application.

As shown in Fig. 5, YAWL adopts the notations of AND-splits/joins and XOR-splits/joins used in WF-nets. Moreover, it introduces *OR-splits* and *OR-joins*. As compared to XOR-splits, which support exclusive choice, OR-splits support multiple choices among conditional branches. Finally, YAWL provides a notation for *removing tokens* from a specified region upon completion of a certain task. This is denoted by associating a dashed lasso to that task that contains the conditions and tasks from which tokens need to be removed or that need to be canceled. This region is called a *cancelation region*, a notion that provides a generalization of the cancelation patterns.

A Running Example: Credit Card Application Process

Let's return to the example credit card application process described earlier in the chapter. To make it more interesting, we extend the process and describe it from the beginning. The process starts when an applicant submits an application (with the proposed amount). Upon receiving an application, a credit clerk checks whether it is complete. If not, the clerk requests additional information and waits until this information is received before proceeding. At the same time, a timer is set so that if

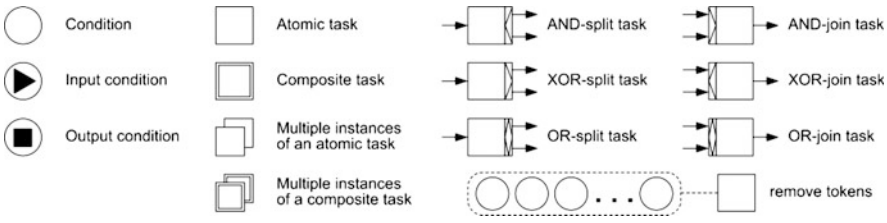

Fig. 5 Modelling elements in YAWL (taken from (van der Aalst & ter Hofstede 2005))

certain period elapses before any information is received, the request for additional information will be sent again. For a complete application, the clerk performs further checks to validate the applicant's income and credit history. Different checks are performed depending on whether the requested loan is large or small. The validated application is then passed on to a manager to decide whether to accept or reject the application. In the case of acceptance, the applicant is notified of the decision and at the same time is asked for his/her preference on any extra features before a credit card is produced and delivered. For a rejected application, the applicant is notified of the rejection and the process ends. Two more facts are to be mentioned in this process. Firstly, an application may be canceled at any time after it was received and before the manager makes the decision. Secondly, for an approved application, three features are offered including customized card, reward program, and secondary cardholders, and any number of them may be chosen.

Figure 6 depicts a YAWL model of the process. We will not go through every element of the model but select a number of typical examples for illustration. Firstly, the task *check for completeness* uses an XOR-split to capture the checking result and an XOR-join to capture further checks to be performed after additional information is received. Next, the place *waiting* models a deferred choice between tasks *receive more info* and *time out*. Thirdly, the selection of extra features is modeled by a subprocess related to the composite task *choose features*. In this subprocess, task *start features* uses an OR-split to capture the fact that a set of extra features can be added, possibly one, two, or all, and task *complete features* uses an OR-join to collect only the features that were actually selected. Note that the definition of a suitable semantics of the OR-join within the context of YAWL can be found in Wynn et al. (2005). Also, task *add secondary cardholders* can have multiple instances, which

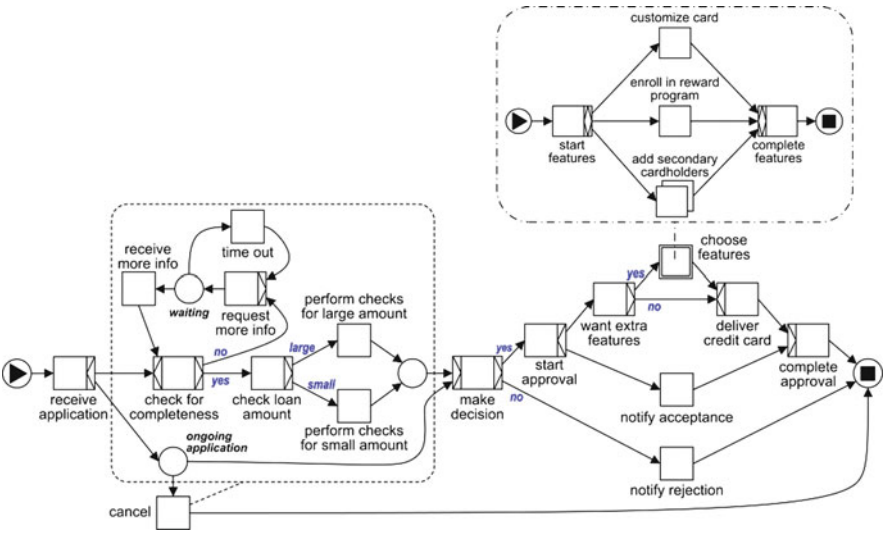

Fig. 6 A credit card application process in YAWL

allow the addition of more than one secondary cardholder in parallel. Finally, in the process model, task *cancel* with its associated cancelation region capture the withdrawal of an ongoing application before an approval/reject decision is made.

In addition to the control-flow definition depicted in Fig. 6, the YAWL model also allows for the specification of the data perspective and the resource perspective. The data specification defines data elements and their usage for exchanging information with the environment, for conditional routing, for creating and synchronizing multiple instances, and so on. Data are represented in XML and data manipulation relies on XML-based standards like XPath and XQuery. The resource perspective specifies task-resource allocation for each task within the process. Note that the term "resource" here refers to human resource, e.g., a role or a participant. Both the data and resource definitions of the process model shown in Fig. 6 will be described further in the section on the YAWL environment later in the chapter.

4 Before Deployment

The development of workflow specifications can be considered as an iterative process, whereby, the specifications are carefully checked and modified to ensure their correctness. In this section, we briefly describe the two techniques, verification and simulation, which can be used to analyze structural and behavioral properties of workflow specifications before deployment. This is followed by a brief description of process configuration, a technique whereby a reference workflow specification is customized based on specific requirements of an organization.

4.1 Verification

Workflow verification is concerned with determining, *in advance*, whether a workflow exhibits certain desirable behaviors. Although one would expect verification functionality to be present in any workflow management system, this is not the case. Typically, these systems at best do some basic syntactical checks but cannot detect the modeling of processes with deadlocks, livelocks, and other anomalies. There are several academic process verification tools. However, until recently, these tools could not verify realistic processes because they assume highly simplified models completely disconnected from real-life languages and systems.

There are established methods for the verification of workflow specifications using Petri nets (van der Aalst 1997). These analysis techniques enable a process designer to answer important questions about a workflow specification, including:

- Can the process model be completed without errors (termination)?
- Are there tasks that are never executed (dead tasks)?
- Are there tasks that are still executing when the process is supposed to be completed (proper completion)?

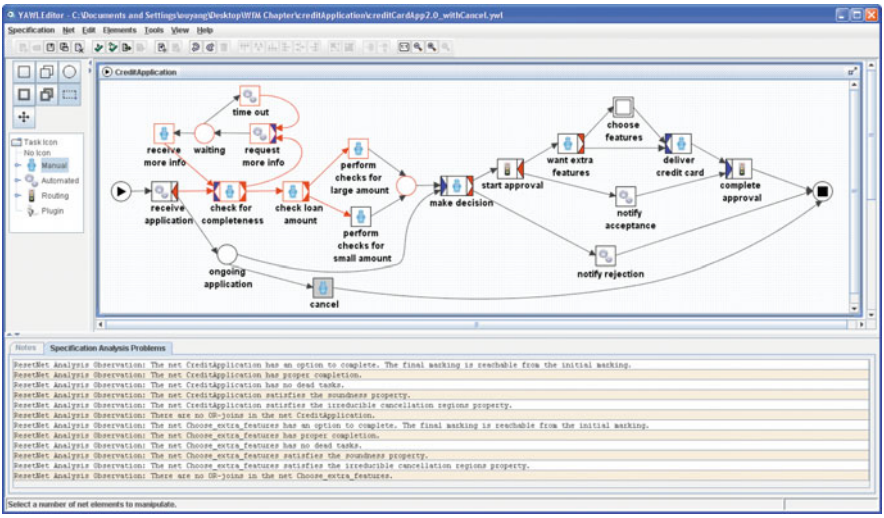

Fig. 7 A screenshot of the verification results for the credit card application process shown in Fig. 6

The answers to these questions are closely related to the soundness property of a workflow specification (van der Aalst 2000).

Three sophisticated verification techniques for workflow specifications with cancelation and OR-joins have been developed in the context of the workflow language YAWL (Verbeek et al. 2007; Wynn et al. 2009c). These techniques make use of Petri nets with reset arcs and Petri nets with inhibitor arcs to detect the soundness property and the relaxed soundness property. Reset arcs can remove tokens from its reset places when a transition fires and they are used to model cancelation regions in YAWL. Similarly, inhibitor arcs can check for empty tokens and they are used to approximate the behavior of OR-joins in YAWL. It is also possible to perform a behavior check of the model using semi-positive transition variants.

Figure 7 shows the verification analysis results for the credit card application example. As this example makes use of complex constructs, such as OR-join and cancelation, to accurately capture the business logic in the workflow specification, no other verification tool except the verification functionality in the YAWL Editor[4] can validate this model and provide information on whether the workflow is sound and whether there are unnecessary OR-joins and/or unnecessary cancelation regions in the specification. In addition to the verification of YAWL process models, Petri nets have also been used to analyze other process modeling languages such as BPMN (Dijkman et al. 2008) and BPEL (Ouyang et al. 2007).

[4]The YAWL editor is a graphical design environment for creating YAWL specifications.

However, there is a clear trade-off between the expressive power of a language (i.e., introducing complex constructs such as cancelation and OR-joins) and ease of verification. As verification relies on the state space analysis, which results in the generation of possible states of a workflow, it is time consuming and can become intractable for large models. Reducing a specification, while preserving its essential properties with respect to a particular analysis problem, is an approach to deal with this complexity. Therefore, a number of soundness preserving reduction rules for Petri nets with reset arcs, and for YAWL elements, are proposed (Wynn et al. 2009a, b).

4.2 Simulation

Workflow simulation enables the analysis of workflow specification with respect to performance metrics such as throughput time, cost, or resource utilization. The main steps in workflow simulation involves developing an accurate simulation model, which reflects the behavior of a process, including the data and resource perspectives, and then performing simulation experiments to better understand the effects of running that process. In general, a simulation model consists of three components: basic model building blocks (e.g., entities, resources, activities, and connectors); activity modeling constructs (e.g., branch, assemble, batch, gate, split, and join); and advanced modeling functions (e.g., attributes, expressions, resource schedules, interruptions, user defined distributions) (Tumay 1996). The interested readers can find more details in van der Aalst et al. (2010), which is dedicated to the topic of business process simulation.

Simulation is regarded as an invaluable tool for process modeling due to its ability to perform quantitative modeling (e.g., cost-benefit analysis and feasibility of alternative designs) as well as stochastic modeling (e.g., external factors and sensitivity analysis) (Giaglis et al. 1996). Simulation has been used for the analysis and design of systems in different application areas and for improving orchestration of supply chain business processes. Simulation can also be used as a decision support tool for business process reengineering to identify bottlenecks and to reduce wait-times between activities. For instance, a simulation model based on the credit card applications process with reliable input data can be used to answer questions about how long it takes on average to process a credit application, how many applications are processed per month, the number of human and nonhuman resources required, the cost of processing these applications, and so on.

Even though simulation is well-known for its ability to assist in long-term planning and strategic decision making, it has not been considered to date as a mainstream technique for operational decision making due to the difficulty of obtaining real-time data in the timely manner to set up the simulation experiments (Reijers 2003; Reijers and van der Aalst 1999). This can be achieved by making closer alignment between a workflow system and a simulation environment, and could involve making use of available case information from workflow system and

historical data from process mining tools (Rozinat et al. 2008; Wynn et al. 2007; Russell et al. 2006a).

The state-of-the-art workflow simulation environment should be powerful enough to fully represent underlying business processes and their environment and should support strategic, as well as operational, decision making. One way to identify the requirements for such a simulation environment could be determined in terms of its support for the control flow patterns (van der Aalst et al. 2003), the data flow patterns (Russell et al. 2005b), and the resource patterns (Russell et al. 2005a). Business process simulation tools survey conducted by Jansen-Vullers and Netjes (Jansen-Vullers and Netjes 2006) highlights the fact that these simulation tools are lacking support for complex control flow patterns as well as many of the data and resource patterns. There is a need for simulation environment to support different resource allocation strategies and resource behaviors. In addition, the simulation environment should offer support for ease of integration of historical data into the experiments. The simulation environment also should provide an ability to add customized attributes. These requirements for a state-of-the-art simulation environment are currently being considered and researched, and there is a proposal to support this as part of the YAWL workflow framework.[5]

4.3 Configuration

A reference model represents a generic business process for a particular domain, which can be customized to realize the business process in an organization. More than one reference model may be available for a particular business domain (e.g., supply chain management), and model selection is a crucial task, which requires a good understanding of available reference models in that domain (Fettke and Loos 2003). Process configuration is concerned with the customization of a process specification based on the different variants of the model by allowing for the enabling or disabling of actions (Gottschalk et al. 2008). To this end, Rosemann and van der Aalst (2007) propose the notion of a *configurable reference modeling language* using Event-Driven Process Chains (EPCs). Although the notion of reference models and the advantage of reusing these models for process design are well known, current approaches for configuring reference process models are manual and thus error-prone (van der Aalst et al. 2008).

It is possible to integrate configuration choices into workflow models as runtime choices. However, the advantage of using the configuration approach is that it allows a clear distinction between configuration choices and runtime choices and results in a smaller and clearer workflow model. The approach proposed in Gottschalk et al. (2008) involves three phases: (1) the build time of the model

[5] www.yawlfoundation.org/theory/simulation.php

when all the variants of a configurable model is specified, (2) the configuration time when a particular workflow variant is selected based on some criteria, and (3) the run time when process instances executed based on the configured model. The authors describe their approach using hiding and blocking operators, and realize the approach in the context of the YAWL language, through Configurable YAWL (C-YAWL). The authors also show the applicability of the approach to other languages such as the workflow engine of SAP R/3 and to BPEL.

For large reference models, the designer can find it difficult to make all the configuration choices one by one. To make this configuration process easier, a questionnaire-based approach is proposed to identify the viability in the reference models and to assist the designer in making configuration decisions (La Rosa et al. 2007, 2009). To ensure the correctness of the resulting configured model, a framework for configuring reference process models in a correctness-preserving manner has been proposed (van der Aalst et al. 2008). The syntactic correctness and the semantic correctness can be checked at each intermediate step of the configuration procedure. If a configuration step violates the constraints, suggestions are provided to make the configuration step correctness-preserving.

5 Dealing with Change

With its roots in office automation and document routing, workflow management systems have traditionally followed an *assembly-line* metaphor, where rigidly structured business processes derive strongly prescriptive process models, which in turn produce invariant process instances. While organizational environments performing highly repetitive activities were early benefactors of workflow solutions, a much larger proportion of workplaces undertake activities that do not easily conform to such constricting representations of their work practices. Due to inflexible modeling frameworks, process models are said to be system-centric, meaning that processes are *straight-jacketed* (van der Aalst et al. 2005) into the paradigm supplied, rather than the paradigm reflecting the way work is actually performed, resulting in often substantial differences between real processes and the models designed to represent them (Rozinat and van der Aalst 2005).

Change is unavoidable in the modern workplace. To remain effective and competitive, organizations must continually adapt their business processes to manage the rapid changes demanded by the dynamic nature of the marketplace or service environment. It is also the case that, even in the most structured processes, deviations or unpredicted events will occur with almost every instantiation. Therefore, so that the benefits of workflow management system may be offered to the broader organizational spectrum, the ability to deal with change must be effectively addressed.

The types of change that workflow systems must deal with are generally categorized into two distinct but related groups: Dynamic Workflow and Exception Handling.

5.1 Dynamic Workflow

Dynamic (or adaptive) workflow refers to the extending of otherwise static workflow processes so that, when change occurs, the process model can be modified or augmented in some way, rather than defaulting to the construction of a completely new model. The change may be considered ad hoc (i.e., only affecting the current instance) or may need to be applied, either temporarily or permanently, to all (or a subset of) current and future instantiations.

Adaptation takes place on two levels. First, the process model is modified, which has associated issues regarding what kinds of changes are allowed and whether the changes maintain support for the objective of the activity. Second, any currently running instances have to be managed when the process model from which it was instantiated changes, which has its own issues, such as whether the instance should be aborted, restarted using the modified model, allowed to continue (so that there are several co-existing versions of the same business process), and other associated problems to do with migration, synchronization, version control, and syntactic and semantic correctness (van der Aalst 2004; Ly et al. 2006; Rinderle et al. 2004). For a closer look at the phenomenon of adaptation, please also refer Hallerbach et al. (2010).

So dynamic workflow provides support for *occasional* changes to the business process model, and assumes the model is basically correct, but incremental or ad hoc changes may be accommodated as required.

An example of a commercial system providing some support for dynamic adaptation is *Tibco iProcess Suite* (version 10.5),[6] which offers an *Orchestrator* component that provides dynamic allocation of subprocess variants at runtime. It requires a construct called a *dynamic event* to be explicitly modeled that contains a number of subprocesses listed as an "array". When execution reaches the dynamic event node, it will execute members of the array based on predefined conditionals, which, like the array, must be statically defined before the process is instantiated – that is, there is no scope for runtime modifications. Another commercial system, COSA (version 5.4),[7] allows *manual* ad hoc runtime adaptations such as reordering, skipping, repeating, postponing, or terminating tasks.

The *ADEPT2* prototype (Reichert et al. 2005) supports process modification during execution (i.e., add, delete, and change the sequence of tasks) both at the model (dynamic evolution) and instance levels (ad hoc changes). Such changes are made to a traditional monolithic model and must be achieved through the manual intervention of an administrator, abstracted to a high-level interaction. The system also supports forward and backward "jumps" through a process instance, but only by authorized staff who instigate the skips manually.

The YAWL system (cf. Sect. 7) provides support for flexibility and dynamic exception handling through the concept of *worklets*, an extensible repertoire of

[6] www.staffware.com/resources/software/bpm/tibco_iprocess_suite_whitepaper.pdf

[7] www.cosa-bpm.com/project/docs/COSA_BPM_5_Productdescription_eng.pdf

self-contained subprocesses and associated selection rules (Adams et al. 2006). This approach directly provides for dynamic change and process evolution without having to resort to off-system intervention and/or system downtime.

5.2 Exception Handling

If an event occurs that impacts on the execution of a process instance but is not explicitly catered for in the process model (such as a process abort, an unavailable resource, or a constraint violation), then certain strategies need to be undertaken to "handle" the event. Traditionally, exceptions are considered to be events that by definition occur rarely. But virtually, every process instance will experience some kind of exception during its execution. It may be that these events are known to occur in a small number of cases, but not often enough to warrant their inclusion in the process model (which implies an off-line, manual handling of such events); or they may be things that were never expected to occur (or maybe never even *imagined* could occur). In any case, when they do happen, since they are not included in the process model, they must be handled in some way before processing can continue. In some cases, the static process model will be modified to capture this unforeseen event, which often involves a large organizational cost (downtime, remodeling, testing, and so on), or in certain circumstances, the entire process must be aborted. However, since most processes are long and complex, neither manual intervention nor process termination is satisfactory solutions (Hagen and Alonso 2000).

Alternately, an attempt might be made to include every possible situation into the process model so that when such events occur, there is a branch in the process to take care of it. This approach often leads to very complex models where much of the original business logic is obscured by exception handling forks, and does not avoid the same problems arising when the next unexpected exception occurs.

Approaches to workflow exception handling generally rely on a high degree of runtime user interactivity, which directly impedes on the basic aim of workflow systems (to bring greater efficiencies to work practices) and distracts users from their primary work tasks into process support activities. For example, most systems support simple deadline expiries (timeouts), but in almost every case, unless an appropriate action is explicitly modeled, a deadline results in a message to an administrator for manual handling.

Russell et al. (2006a) present a framework for the classification of exception handling in process-aware information systems based on patterns. They point out that systems supporting some degree of exception handling may allow exceptions to occur during the execution of a process instance, then provide mechanisms called *exception-handlers* (external to, but linked to, the "parent" business process) to handle the exception and allow the process instance to continue unhindered. These handlers may be defined graphically, or as rules, or as a combination of the two. Thus, a distinction between static workflow systems and exception handling systems is

that in the former, all business rules, conditions, and exception handling branches must be explicitly defined in the business process model itself, whereas for the latter, the exception handling parts of the process can be separated from the main business process. It is important to note that, typically, handlers can only be specified for exceptions that are expected (because the definition of exception-handlers must be completed before an instance is executed), although some recent developments in this field also provide the ability to capture and handle *unexpected* exceptions at runtime (for example, the YAWL *Worklet Service* (Adams et al. 2007)).

For any work process, it may be more productive to accept the fact that deviations to any plan will occur in practice and to implement support mechanisms, which allow for those behaviors to be *implicitly* incorporated into the model, rather than to develop a closed system that tries to anticipate all possible events, then fails to accommodate others that (inevitably) occur. This notion supports the idea of evolutionary workflow support systems, which over time and through experience *tune* themselves to the business process they are supporting.

6 Beyond Enactment

When an instance of a workflow specification is being executed, workflow participants can monitor its progress. Also, historical information about the execution of the various workflow instances is saved by the workflow system. This information can be used for several purposes, e.g., process mining and workflow recovery. In this section, we briefly discuss the topics of workflow monitoring and process mining.

6.1 Monitoring and Escalation

Active workflow monitoring enables workflow administrators to be aware of workflows, which are deadlocked, taking exceptionally long time to complete, etc. With workflow systems typically handling long-running business processes, the need to monitor these processes and to act quickly when changes are required is paramount. However, it is typically not possible or easy to change a deployed workflow. These situations become more and more unavoidable at runtime due to the nature of interorganization workflows. In such situations, there is a need to consider escalation strategies, which involve making decisions regarding alternative arrangements to achieve the goal of completing the workflow within a reasonable timeframe. Escalation may imply "performing a task in a different way, allowing less qualified people to do certain tasks, or making decisions based on incomplete data" (van der Aalst et al. 2007b). van der Aalst et al. (2007b) propose a set of escalation strategies by looking at the three perspectives of workflow. They include alternative path, escalation subprocess, task predispatching, overlapping and prioritization for the

process perspective, resource redeployment and batching for the resource perspective, and deferred data gathering and data degradation for the data perspective.

6.2 Process Mining

Process mining is concerned with discovering, monitoring, and improving business process by extracting relevant information from the event logs produced by a wide variety of systems (van der Aalst et al. 2004b; Weijters et al. 2007). The basic idea behind process mining is to learn from observable execution behavior of a business process by analyzing event logs, audit trails, and transaction logs, which may contain detailed information about the activities of the business processes that have been executed (van der Aalst et al. 2007a).

A wide range of process mining techniques and algorithms exist to perform analysis on the control, the data, and the resourcing perspectives of a workflow specification. The research group headed by Prof. Wil van der Aalst has been actively researching in the area of process mining for a number of years (http://www.processmining.org). To support this research, the open-source Process Mining ProM framework has been developed. ProM supports a pluggable software architecture, which allows developers and analysts to add their own process mining techniques with ease. ProM currently offers almost 200 plug-ins. Over the last couple of years, ProM has been applied in a wide range of real-life case studies, and several ideas have been incorporated in the commercial tools such as ARIS and the BPM suite of Pallas Athena (van der Aalst et al. 2007a).

7 A Sample System: The YAWL Environment

Today, many workflow management systems are available, both commercial and open source. Firstly, let's have a brief look at a number of commercial products. *Staffware* is one of the leading workflow systems since 1998 and is now owned by TIBCO Software. *COSA* is a Petri-net-based workflow system developed by a German company called Ley GmbH. *SAP R/3 Workflow* is an integrated workflow component of SAP R/3 software suite and now runs over the platform of SAP NetWeaver. *Visual WorkFlo*, part of the FileNet's Panagon suite (Panagon WorkFlo Services), is one of the oldest and best established products on the market of the workflow industry. *WebSphere MQ Workflow* is developed by IBM for process automation and enables use with WebSphere Business Integration Modeler and Monitor for design, analysis, simulation, and monitoring of process improvements. *Oracle BPEL Process Manager*, now part of the Oracle SOA Suite, is a BPEL engine that enables enterprises to orchestrate disparate applications and Web services into business processes.

In the area of open source workflow systems, the four most downloaded systems (as at July 2008) are OpenWFE, jBPM, Enhydra Shark, and YAWL. OpenWFE

(or more precisely, OpenWFEru or Ruote) is a workflow management system written in Ruby. It is aimed for developers and distributed under the BSD License. JBoss jBPM is abbreviation of Java for Business Process Management. It is JBoss' (RedHat's) workflow management system and is written in Java. The tool is distributed through SourceForge under the LGPL license. Enhydra Shark is a Java workflow engine offering from Together Teamlösungen and ObjectWeb. While it is an open source offering, its architecture allows for the use of closed-source or proprietary plug-ins to enhance it. The open-source version of Enhydra Shark is licensed according to the LGPL. Finally, the YAWL System (van der Aalst et al. 2004a) and its environment represent an implementation of a workflow management system supporting the YAWL language. Like jBPM, the YAWL system is distributed through SourceForge under the LGPL license. The YAWL environment is unique in its near-complete support for the workflow patterns. It is therefore used as a sample workflow management system for discussion in this section.

7.1 Architecture

The high-level architecture of the YAWL environment is depicted in Fig. 8. The most obvious feature of the environment is the separation of functionality between the core YAWL Workflow Engine and a number of so-called YAWL Custom

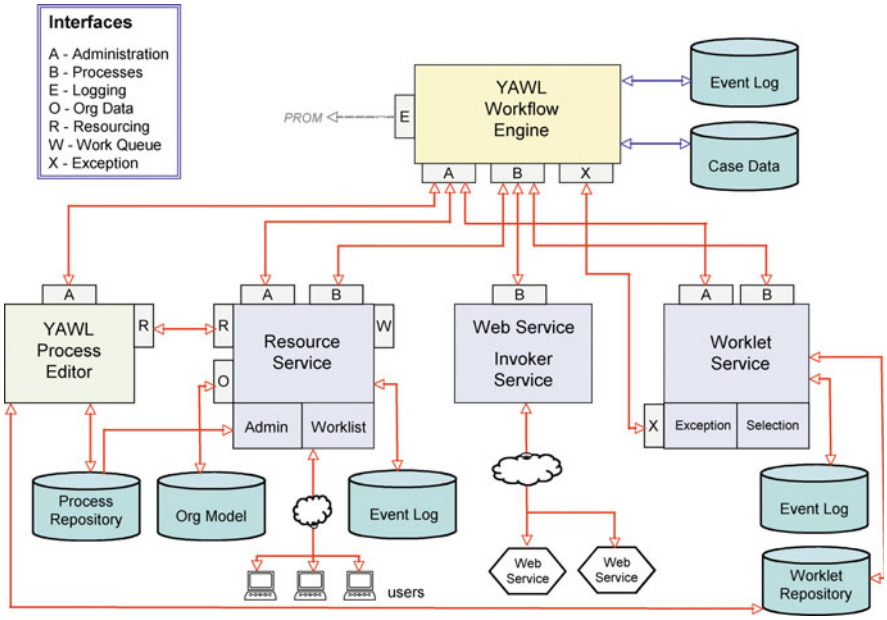

Fig. 8 YAWL system architecture

Services. Inspired by the "web services" paradigm, end-users, applications, and organizations are all abstracted as services in YAWL. Figure 8 shows the three major YAWL services: (1) the Resource Service, with integrated worklist handler and administration tool; (2) the Web Services Invoker; and (3) the Worklet Service, which provides dynamic flexibility and exception handling capabilities.

Workflow specifications are designed using the YAWL Process Editor and stored in the repository as XML. From there, they may be deployed into the YAWL Engine, which, after performing all necessary verifications and task registrations, makes the specifications available to the environment so that they can be instantiated through the Engine, leading to workflow instances. The Engine handles the execution of these cases, and based on the state of a case and its specification, the Engine determines which tasks and events it should offer to the environment.

YAWL Custom Services interact with the engine and each other via a number of interfaces, which provide methods for object and data passing via HTTP requests and responses. All data are passed as XML; objects are marshaled into XML representations on the server side of each interface and reconstructed back to objects on the client side. The YAWL Engine provides four interfaces:

- Interface A: which provides endpoints for process definition, administration, and monitoring;
- Interface B: which provides endpoints for client and invoked applications and workflow interoperability, and is used by services to connect to the engine, to start and cancel case instances, and to check workitems in and out of the engine;
- Interface E: which provides access to archival data in the engine's process logs; and
- Interface X: which allows the engine to notify custom services of certain events and checkpoints during the execution of each process instance where process exceptions either may have occurred or should be tested for.

The YAWL interfaces correlate somewhat loosely to those defined in the Workflow Reference Model (WRM) of the Workflow Management Coalition (WfMC) (Hollingsworth 1995). The WRM describes a core Workflow Enactment Service (or Engine) interacting with a number of generic components via a defined set of standardized interfaces and data interchange formats. In addition to the core Engine, the Workflow Reference Model identifies five major component types and their interfaces. YAWL's interface A corresponds strongly to the WRM interface 1 (and partially to interface 5), while YAWL's interface B relates to WRM interfaces 2, 3, and 4. YAWL interface E corresponds to parts of WRM interface 5 also.

The YAWL Resource Service incorporates a full-featured worklist handler and administration toolset, implemented as a series of web pages. The service automatically assigns tasks to resources and places them in the appropriate work queues based on design time specifications and runtime decisions, while the administration tools can be used to manually control workflow instances (e.g., loading or removing a workflow specification, launching, or canceling case instances), manage resources and allocate them to tasks, and provide information about the state of running workflow instances.

The Resource Service provides three additional interfaces that allow developers to implement other worklist handlers and administration tools while leveraging the full functionality of the service. Interface R provides organizational data to (authorized) external entities such as the YAWL Process Editor; Interface W provides access to the internal work queue routing functionalities; and Interface O allows organizational data to be provided to the service from any data source. In addition, the service's framework is fully extendible, allowing further constraints, filters, and allocation strategies to be "plugged in" by developers.

The worklist handler, incorporated into the Resource Service, corresponds to the classical worklist handler present in most workflow management systems. It is the component used to assign work to users of the system. Through the worklist handler, users are offered and allocated work items, and can start and signal their completion. In traditional workflow systems, the worklist handler is embedded in the workflow engine. In YAWL, however, it is considered to be a service completely decoupled from the engine so that the Engine has no knowledge of how work will be assigned.

The YAWL Web Services Invoker is the glue between the engine and other web services. Note that it is unlikely that web services will be able to directly connect to the YAWL engine, since they will typically be designed for more general purposes than just interacting with a workflow engine. Similarly, it is desirable not to adapt the interface of the engine to suit specific services; otherwise, this interface will need to cater for an undetermined number of message types. Accordingly, the YAWL web services broker acts as a mediator between the YAWL engine and external web services that may be invoked by the engine to delegate tasks (e.g., delegating a "payment" task to an online payment service).

The YAWL Worklet Service (Adams et al. 2006, 2007) comprises two discrete but complementary subservices: a Selection Service, which enables dynamic flexibility for otherwise static process instances, and an Exception Service, which provides facilities to handle both expected and unexpected process exceptions (i.e., events that may happen during the lifecycle of a process instance that affect the execution of the instance but were not explicitly modeled in the process specification) at runtime.

In addition to the three services shown in Fig. 8, any number of additional custom services can be implemented for particular interaction purposes with the YAWL Engine. For example, a custom YAWL service could offer communication with devices such as mobile phones, printers, and assembly robots. A custom service may be used to manipulate the data of certain tasks, or may be implemented to enhance the presentation of work to end-users (for example, via a graphical interface or as a component within a virtual environment). It is also possible that there are concurrent multiple services of the same type, e.g., multiple worklist handlers, web services brokers, and exception handling services. For example, there may exist multiple implementations of worklist handlers (for example, customized for a specific application domain or organization) and the same worklist handler may be instantiated multiple times (for example, one worklist handler per geographical region).

7.2 Design Time

A YAWL workflow with control, data, and resource perspectives can be created using the YAWL Process Editor, which is a standalone component of the YAWL workflow system. Figure 9 provides a screenshot of the credit card application process modeled in the YAWL Editor. The control flow perspective of the workflow is specified using the YAWL icons on the top-left side of the screen. The data perspective of the workflow such as input and output data as well as the data used for flow decisions (XOR-split and OR-splits) are modeled using XML data elements. The resource perspective specifies who should do a particular task from a set of available resources from the organizational database. This feature requires client-server access to an executing resource service via interface R (Fig. 8) so that information regarding resources can be retrieved, and can be configured in the YAWL Editor using a 5-step wizard. As an example, Fig. 10 shows a screenshot of the second step in specifying the resource perspective for task *make decision* in the credit card application process.

The YAWL specification can be checked using the "Validate Specification" feature to ensure the structural correctness of the workflow. Furthermore, the specification can be analyzed using the "Analyze Specification" feature to ensure the behavioral correctness of the workflow with regards to the control flow. A validated specification can then be exported to the YAWL engine for enactment.

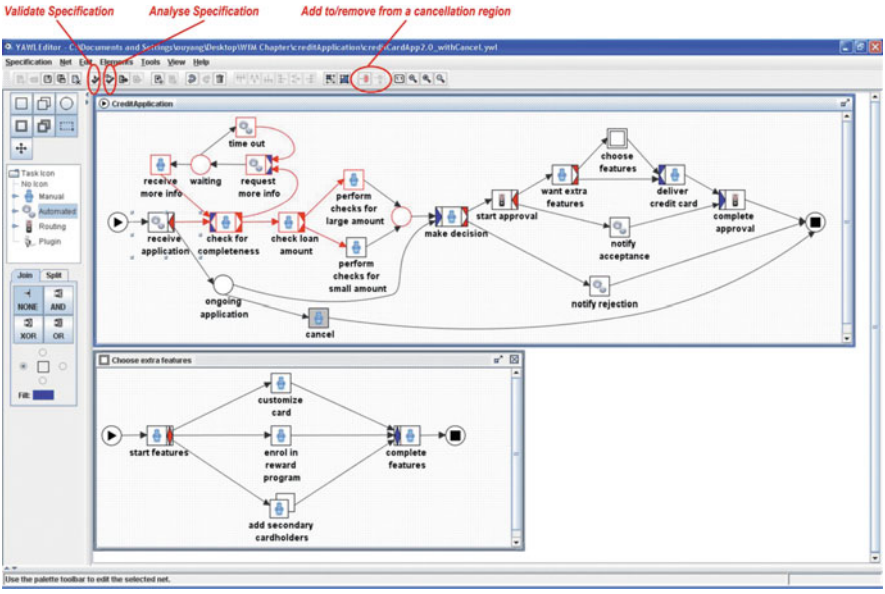

Fig. 9 Using the YAWL Editor for specifying the control flow perspective of the credit card application process shown in Fig. 6

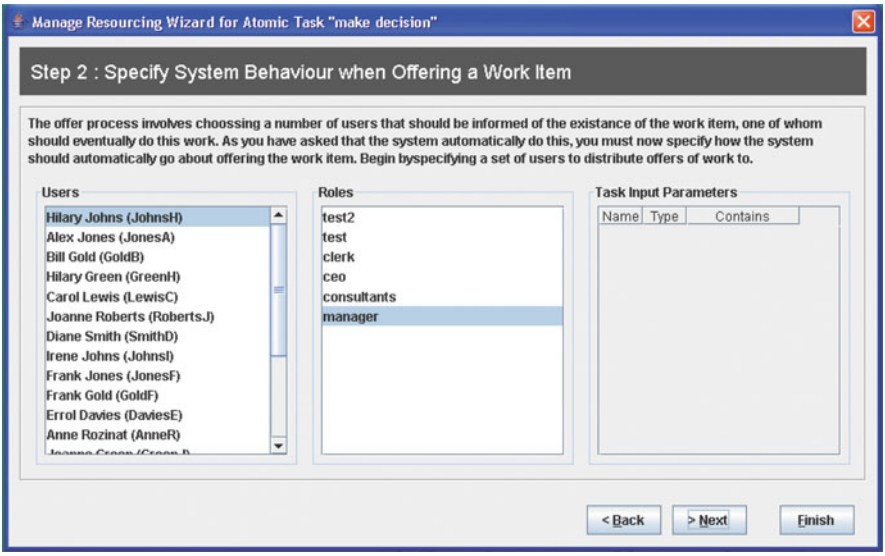

Fig. 10 Using the YAWL Editor for specifying the resource perspective (Task *make decision* should be performed by a user with a manager role)

Currently, version 2.0 of the YAWL Editor provides support for specifying extended attributes such as cost, priority, etc., integrated support for timeout tasks using timers, and the support for 38 out of 43 resource patterns. It can be downloaded from SourceForge.[8]

7.3 Runtime

At runtime, the YAWL Engine presents events and tasks to the environment as they occur during the lifecycle of process instantiations via the interfaces described earlier. Using those interfaces, custom services may elect to be notified of certain events (i.e., when a workitem becomes enabled, or is canceled, or when a case instance completes) or of changes in the status of existing workitems and case instances.

For example, on receiving notification from the Engine of an item-enabled event (i.e., when a work item becomes ready for execution), a custom service may elect to "check-out" the workitem from the Engine. On doing so, the Engine marks the work item as *executing* and effectively passes operational control for the work item to the custom service. When the service has finished processing the work item, it will check it back into the Engine, at which point the Engine will mark the work item as *completed* and proceed with process execution.

[8] http://sourceforge.net/projects/yawl/

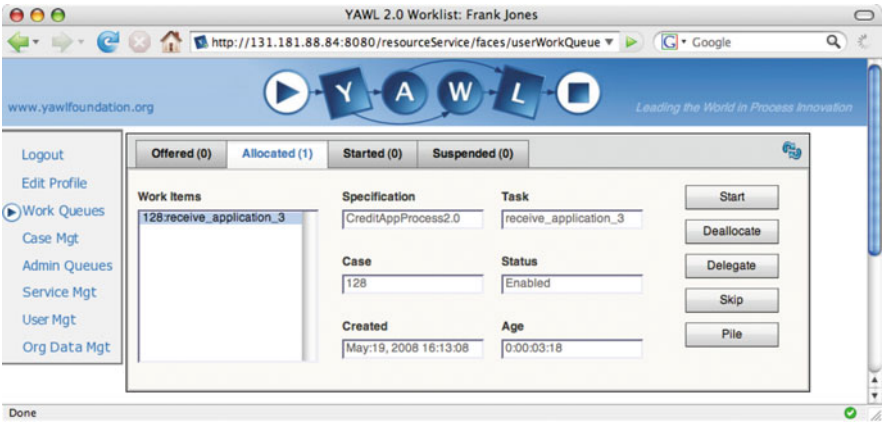

Fig. 11 The YAWL Work Queues (allocated queue active)

An example of such a service is the Resource Service, which provides as a component a worklist handler for the execution, updating, and completion of work items at runtime. The user interface is provided as a series of web pages. Each work item is presented to the appropriate user work queues based on four interaction points: offered, allocated, started, and suspended.

Figure 11 shows a screen of an allocated work queue in the YAWL runtime environment. The screen displays the information about a work item that has been allocated, including the process specification, the identifier of the process instance (i.e., case number), the task to which the work item belongs, and its creation time and age. There are also functionalities that support different operations with the work item, for example, to delegate the work item to another human resource. The types of functionality available vary relevant to each of the four work queues.

When a work item is started, its data may be viewed and edited via a dynamically generated form. Each completed work item is passed back to the Engine, allowing the case instance to progress. While the Resource Service offers a default worklist handler, custom services may be designed to handle the work and events offered by the YAWL Engine in a variety of ways. For example, the exception-handling component of the Worklet Service uses the same task and event notifications to determine if exceptions have occurred during a process instance's life cycle and take appropriate action as required.

8 A Case Study: YAWL4Film

As part of the Australian Research Council Centre of Excellence for Creative Industries and Innovation,[9] we move well beyond the traditional use of workflow management systems and investigate how they can deliver benefits to the field of

[9] http://www.cci.edu.au

screen business. The screen business comprises all creative and business related aspects and processes of film, television, and new media content, from concept to production and then distribution. A film production process includes daily shooting activities like acting, camera, and sound recording over a period varying from days to years. It involves handling large amounts of forms and reports on a daily basis and coordinating geographically distributed stakeholders. Traditionally, the forms and reports are purely paper-based and the production of these documents is a highly manual process. Not surprisingly, such a process is time-consuming and error-prone, and can easily increase the risk of delays in the schedule.

Within the above context, YAWL was applied to the automation of film production processes (Ouyang et al. 2008a, b). This led to the development of a prototype, namely YAWL4Film, which exploits the principles of workflow in order to coordinate work distribution with production teams, automate the daily document processing and report generation, ensure data synchronization across distributed nodes, archive and manage all shooting related documents systematically, and document experiences gained in a film production project for reuse in the future. The system was successfully deployed in two pilot projects at the Australian Film, Television, and Radio School in October 2007.

Below, we briefly describe YAWL4Film. It consists of a YAWL model capturing the control-flow, data, and resource perspectives of a film production process. It also extends the general YAWL system with customized user interface to support templates used in professional filmmaking.

8.1 Process Model

Figure 12 depicts the YAWL model of a film production process. An instance of the process model starts with the collection of specific production documents (e.g., *cast list*, *crew list*, *location notes*, and *shooting schedule*) generated during the preproduction phase. Next, the shooting starts and is carried out on a daily basis. Each day, tasks are performed along two main parallel streams. One stream focuses on the production of a *call sheet*. It starts from task *Begin Call Sheet* and ends with task *Finish Call Sheet*. A call sheet is a daily shooting schedule. It is usually maintained by the production office and is sent out to all cast and crew the day prior. A draft call sheet can be created from the shooting schedule. It may go through any number of revisions before it is finalized, and most of the revisions result from the changes to the shooting schedule. The other stream specifies the flow of onset shooting activities and supports the production of a *daily process report* (DPR). It starts with task *Kick Off on-set* and ends with task *Distribute DPR*. At first, tasks are executed to record the logs and technical notes about individual shooting activities into a number of documents. These are *continuity log* and *continuity daily*, which are filled by the Continuity person, *sound sheet* by a Sound Recordist, *camera sheet* by a Camera Assistant, and *2nd Assistant Director (AD) Report* by the 2nd AD. It is possible to interrupt filling in the continuity log and the 2nd AD report, e.g., for a

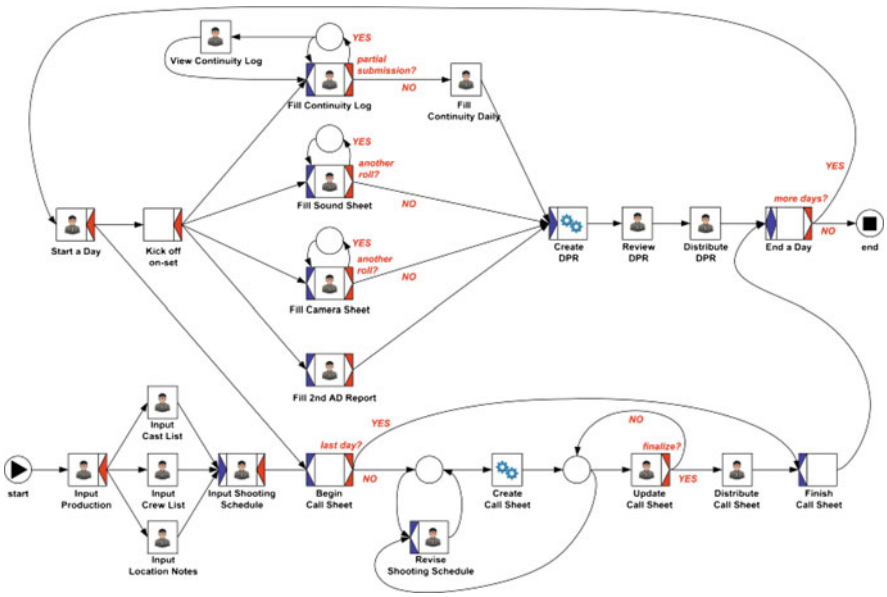

Fig. 12 A film production process model in YAWL

meal break, and then to resume the work after the break. Also, there can be many camera and sound sheets to be filled in during a shooting day. Upon completion of these on-set documents, a DPR can be generated and passed onto the Production Manager for review. After the review, the DPR is circulated to certain crew members, e.g., Producer and Executive Producer.

In this process model, it is interesting to see how the OR-join associated with task *End a Day* behaves. Before the first shooting day starts, an instance of the call sheet branch is executed for producing the first day's call sheet. Since it is the only active incoming branch to task *End a Day*, the task will be performed once the call sheet has completed, without waiting for the completion of a DPR. In this case, the OR-join behaves like an XOR-join. On the other hand, if both call sheet and DPR branches are active (which is the case for the rest of the shooting days), the OR-join behaves like an AND-join.

8.2 User Interface

Most tasks in the film production process are manual (annotated with an icon of a human) and require users to fill in forms. While the YAWL environment supports automatic generation of screens based on input/output parameters and their types, in order to support templates used in professional filmmaking, custom-made Web forms were created and linked to the worklist handler of YAWL. Figure 13 for

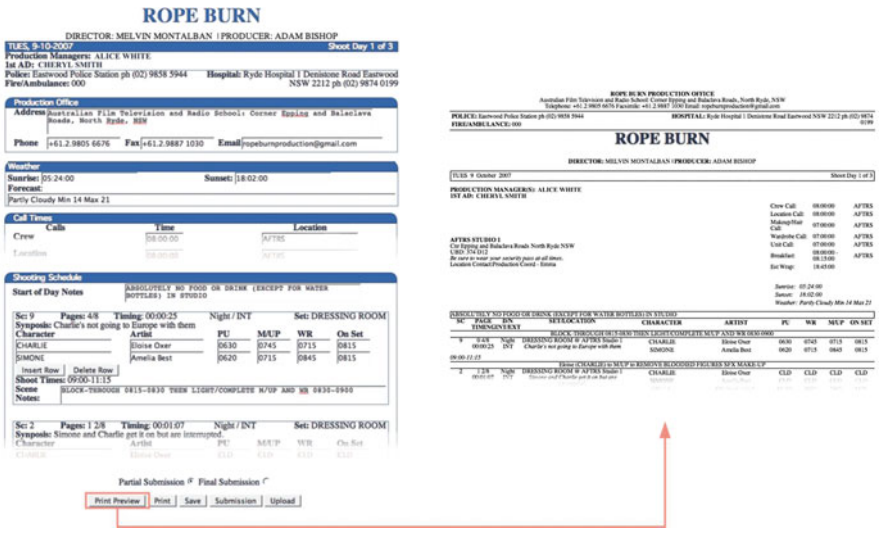

Fig. 13 An example of custom Web form – call sheet

example depicts the Web form for task *Update Call Sheet* (in Fig. 12) as seen by a production office crew member. The custom forms and their links to YAWL were developed using standard Java technology. Each form can load/save an XML file (complying with the schema of the work item), and submit the form back to the worklist handler once it has been completed by the user. Upon submission, a backup copy is stored on the server. Moreover, each form provides data validation upon save and submission to prevent the generation of invalid XML documents that would block the execution of the process. Finally, a print-preview function[10] allows the user to generate a printer-ready document from the Web form, which resembles the hard copy format used in practice in this business.

9 Outlook

This chapter covered many of the main areas that are of relevance in modern workflow management, and more broadly, modern Business Process Management. These included workflow patterns, which are part of the conceptual foundations of workflow management, a number of workflow languages, which exhibit different approaches to workflow specification, and more advanced topics such as handling changes and unexpected exceptions, simulation, verification, and configuration. An existing workflow management system was presented in order to demonstrate some state-of-the-art aspects of workflow management. However, space considerations

[10]This function relies on XSLT transformations to convert the XML of the form to HTML.

prevented in-depth treatment of the various topics covered, and some topics were not covered at all, e.g., support at the language level for interprocess communication (Aldred et al. 2007; Decker and Barros 2007). Nonetheless, we hope that enough pointers were provided to the reader for further study or exploration.

References

Adams M, ter Hofstede AHM, Edmond D, van der Aalst WMP (2006) Worklets: a service-oriented implementation of dynamic flexibility in workflows. In: Proceedings of the 14th international conference on cooperative information systems. LNCS, vol 4275. Springer, Berlin, pp 291–308

Adams M, ter Hofstede AHM, van der Aalst WMP, Edmond D (2007) Dynamic, extensible and context-aware exception handling for workflows. In: Proceedings of the 15th international conference on cooperative information systems. LNCS, vol 4803, pp 95–112

Aldred L, van der Aalst WMP, Dumas M, ter Hofstede AHM (2007) Communication abstractions for distributed business processes. In: Proceedings of the 19th international conference on advanced information systems engineering. LNCS, vol 4495. Springer, pp 409–423

Decker G, Barros AP (2007) Interaction modeling using BPMN. In: Proceedings of business process management workshops 2007. LNCS, vol 4928. Springer, Heidelberg, pp 208–219

Dijkman R, Dumas M, Ouyang C (2008) Semantics and analysis of business process models in BPMN. Inform Softw Tech 50(2):1281–1294

Fettke P, Loos P (2003) Classification of reference models – a methodology and its application. Inform Syst E Bus Manage 1(1):35–53

Fischer L (ed) (2005) Workflow handbook 2005. Workflow Management Coalition

Giaglis G, Paul R, Doukidis G (1996) Simulation for intra- and inter-organisational business process modeling. In: Proceedings of the 28th conference on winter simulation, pp 1297–1308

Gottschalk F, van der Aalst WMP, Jansen-Vullers M, La Rosa M (2008) Configurable workflow models. Int J Cooper Inform Syst 17(2):177–221

Hagen C, Alonso G (2000) Exception handling in workflow management systems. IEEE Trans Softw Eng 26(10):943–958

Hallerbach A, Bauer T, Reichert M (2010) Configuration and management of process variants. In: vom Brocke J, Rosemann M (eds) Handbook on business process management, vol 1. Springer, Heidelberg

Hollingsworth D (1995) The workflow reference model. http://www.wfmc.org/standards/docs/tc003v11.pdf. Accessed 4 Apr 2008

Jablonski S, Bussler C (1996) Workflow-management: modeling concepts, architecture and implementation. International Thomson Computer Press, London

Jansen-Vullers M, Netjes M (2006) Business process simulation – a tool survey. In: Proceedings of the 7th workshop and tutorial on practical use of coloured petri nets and the CPN tools, Aarhus, Denmark

La Rosa M, Lux J, Seidel S, Dumas M, ter Hofstede AHM (2007) Questionnaire-driven configuration of reference process models. In: Proceedings of the 19th international conference on advanced information systems engineering. LNCS, vol 4495. Springer, pp 424–438

La Rosa M, van der Aalst WMP, Dumas M, ter Hofstede AHM (2009) Questionnaire-based variability modeling for system configuration. Software Syst Model 8(2):251–274

Ly LT, Rinderle S, Dadam P (2006) Semantic correctness in adaptive process management systems. In: Proceedings of the 4th international conference on business process management. LNCS, vol 4102. Springer, pp 193–208

Murata T (1989) Petri nets: properties, analysis and applications. Proc IEEE 77(4):541–580

Ouyang C, Verbeek HMW, van der Aalst WMP, Breutel S, Dumas M, ter Hofstede AHM (2007) Formal semantics and analysis of control flow in WS-BPEL. Sci Comput Program 67(2–3):125–332

Ouyang C, ter Hofstede AHM, La Rosa M, Rosemann M, Shortland K, Court D (2008a) Camera, set, action: automating film production via business process management. In: Proceedings of the CCI conference on creating value: between commerce and commons, the ARC Centre of Excellence for the Creative Industries and Innovation, Brisbane, Australia

Ouyang C, La Rosa M, ter Hofstede AHM, Shortland K (2008b) Towards Web-scale workflows for film production. IEEE Internet Comput 12(5):53–61

Ouyang C, Dumas M, van der Aalst WMP, ter Hofstede AHM, Mendling J (2009) From business process models to process-oriented software systems. ACM Trans Softw Eng Meth 19(1): Article No. 2

Reichert M, Rinderle S, Kreher U, Dadam P (2005) Adaptive process management with ADEPT2. In: Proceedings of the 21st international conference on data engineering, Tokyo, Japan, pp 1113–1114

Reijers HA (2003) Design and control of workflow processes: business process management for the service industry. LNCS, vol 2617. Springer, Berlin

Reijers HA, van der Aalst WMP (1999) Short-term simulation: bridging the gap between operational control and strategic decision making. In: Proceedings of the IASTED conference on modeling and simulation, Philadelphia, USA, pp 417–421

Rinderle S, Reichert M, Dadam P (2004) On dealing with structural conflicts between process type and instance changes. In: Proceedings of the 2nd international conference on business process management. LNCS, vol 3080, pp 274–289

Rosemann M, van der Aalst WMP (2007) A configurable reference modeling language. Inf Syst 32(1):1–23

Rozinat A, van der Aalst WMP (2005) Conformance testing: measuring the fit and appropriateness of event logs and process models. In: Proceedings of BPM 2005 workshops (workshop on business process intelligence). LNCS, vol 3812. Springer, Berlin, pp 163–176

Rozinat A, Wynn MT, van der Aalst WMP, ter Hofstede AHM, Fidge CJ (2008) Workflow simulation for operational decision support using design, historic and state information. In: Proceedings of the 6th international conference on business process management. LNCS, vol 5240. Springer, Heidelberg, pp 196–211

Russell N, van der Aalst WMP, ter Hofstede AHM, Edmond D (2005a) Workflow resource patterns: Identification, representation and tool support. In: Proceedings of the 17th conference on advanced information systems engineering. LNCS, vol 3520. Springer, pp 216–232

Russell N, ter Hofstede AHM, Edmond D, van der Aalst WMP (2005b) Workflow data patterns: Identification, representation and tool support. In: Proceedings of the 24th international conference on conceptual modeling. LNCS, vol 3716. Springer, Berlin, pp 353–368

Russell N, van der Aalst WMP, ter Hofstede AHM (2006a) Workflow exception patterns. In: Proceedings of the 18th international conference on advanced information systems engineering. LNCS, vol 5074. Springer, Berlin, pp 288–302

Russell N, ter Hofstede AHM, van der Aalst WMP, Mulyar N (2006b) Workflow control-flow patterns: a revised view. BPM Center Report BPM-06-22, BPMcenter.org

Tumay K (1996) Business process simulation. In: Proceedings of the 28th conference on winter simulation, pp 93–98

van der Aalst WMP (1997) Verification of workflow nets. In: Proceedings of the 18th application and theory of Petri nets. LNCS, vol 1248. Springer, Heidelberg, pp 407–426

van der Aalst WMP (2000) Workflow verification: finding control-flow errors using Petri net-based techniques. In: Business process management: models, techniques and empirical studies. LNCS, vol 1806. Springer, pp 19–128

van der Aalst WMP (2004) Exterminating the dynamic change bug: a concrete approach to support workflow change. Inform Syst Front 3(3):297–317

van der Aalst WMP, ter Hofstede AHM (2005) YAWL: yet another workflow language. Inf Syst 30(4):245–275

van der Aalst WMP, van Hee KM (2002) Workflow management: models, methods, and systems. MIT press, Cambridge, MA

van der Aalst WMP, ter Hofstede AHM, Kiepuszewski B, Barros AP (2003) Workflow patterns. Distrib Parallel Database 14(3):5–51

van der Aalst WMP, Aldred L, Dumas M, ter Hofstede AHM (2004a) Design and implementation of the YAWL system. In: Proceedings of the 16th international conference on ad-vanced information systems engineering. LNCS, vol 3084. Springer, Heidelberg, pp 142–159

van der Aalst WMP, Weijters AJMM, Maruster L (2004b) Workflow mining: discovering process models from event logs. IEEE Trans Knowl Data Eng 16(9):1128–1142

van der Aalst WMP, Weske M, Grünbauer D (2005) Case handling: a new paradigm for business process support. Data Knowl Eng 53(2):129–162

van der Aalst WMP, van den Brand PCW, van Dongen BF, Günther CW, Mans RS, Alves de Medeiros AK, Rozinat A, Song M, Verbeek HMW, Weijters AJMM (2007a) Business process analysis with ProM. In: Proceeding of the 17th annual workshop on information technologies and systems, pp 223–224

van der Aalst WMP, Rosemann M, Dumas M (2007b) Deadline-based escalation in process-aware information systems. Decis Support Syst 43(2):492–511

van der Aalst WMP, Dumas M, Gottschalk F, ter Hofstede AHM, La Rosa M, Mendling J (2008) Correctness-preserving configuration of business process models. In: Proceedings of funda-mental approaches to software engineering. LNCS, vol 4961. Springer, pp 46–61

van der Aalst WMP, Nakatumba J, Rozinat A, Russell N (2010) Business process simulation. In: vom Brocke J, Rosemann M (eds) Handbook on business process management, vol 1. Springer, Heidelberg

Verbeek HMW, van der Aalst WMP, ter Hofstede AHM (2007) Verifying workflows with cancelation regions and OR-joins: an approach based on relaxed soundness and invariants. Comput J 50(3):294–314

Weijters AJMM, van der Aalst WMP, van Dongen B, Günther C, Mans R, Alves de Medeiros AK, Rozinat A, Song M, Verbeek HMW (2007) Process mining with ProM. In: Proceedings of the 19th Belgium–Netherlands conference on artificial intelligence, Utrecht, The Netherlands

Weske M (2007) Business process management: concepts, languages, architectures. Springer, Berlin

White S (2004) Process modeling notations and workflow patterns. BPTrends, pp 1–24

Wynn MT, Edmond D, van der Aalst WMP, ter Hofstede AHM (2005) Achieving a general, formal and decidable approach to the OR-join in workflow using reset nets. In: Proceedings of the 26th international conference on application and theory of petri nets and other models of concurrency. LNCS, vol 3536. Springer, Heidelberg, pp 423–443

Wynn MT, van der Aalst WMP, ter Hofstede AHM, Edmond D (2006) Verifying workflows with cancelation regions and OR-joins: an approach based on reset nets and reachability analysis. In: Proceedings of the 4th international conference on business process management. LNCS, vol 4102. Springer, Berlin, pp 389–394

Wynn MT, Dumas M, Fidge CJ, ter Hofstede AHM, van der Aalst WMP (2007) Business process simulation for operational decision support. In: Proceedings of the 3rd international workshop on business process intelligence. LNCS, vol 4928. Springer, Berlin, pp 66–77

Wynn MT, Verbeek HMW, van der Aalst WMP, ter Hofstede AHM, Edmond D (2009a) Reduction rules for reset workflow nets. Inf Sci 179(6):769–790

Wynn MT, Verbeek HMW, van der Aalst WMP, ter Hofstede AHM, Edmond D (2009b) Reduction rules for workflows with cancelation regions and OR-joins. Inform Softw Tech 51 (6):1010–1020

Wynn MT, Verbeek HMW, van der Aalst WMP, ter Hofstede AHM, Edmond D (2009c) Business process verification – finally a reality! Bus Process Manage J 15(1):74–92