

Workflow Management in an Internet Environment

Martin Waardenburg
Maarten van Emmerik

Web-based systems for workplace and facility automation are rapidly gaining in popularity because they enable organizations to share data and business processes with customers, partners and suppliers. Typically data is stored in a relational database and users can access information via an application server which includes rules and business processes. Since business processes may change frequently they should not be statically programmed on the server but instead users should be able to dynamically design and modify workflows behind these processes. Many workflow management systems are available today but they often lack true integration with a Web-based environment or do not support complex workflow patterns. Axxerion is designed to address these limitations by integrating high-end workflow technology based on Petri Nets with a fully web-based graphical environment for workflow design and visualization.

Workflow Management

Tools for customizing business processes are typically referred to as workflow management systems. A workflow management system assigns work, passes it on and tracks its progress. It can be a standalone solution that interfaces with other products or can be embedded in an enterprise application. Workflow management is still a relatively new field in computing science. Despite the efforts of the Workflow Management Coalition (WfMC, [1]), workflow management systems use a large variety of languages and concepts based on different paradigms. There is currently no generally accepted standard for describing workflows [2]. To be able to compare various workflow management systems, a comprehensive set of workflow patterns has been defined in [2]. The result of this research reveals that the expressive power of contemporary systems leaves much to be desired, and that these systems support very different sets of patterns.

Furthermore, the workflow definitions used by many systems are often not formally defined which makes it hard to predict the behavior of these workflows.

When selecting a workflow management system, it makes sense to focus on workflow management systems that are based on a formal foundation and provide support for the workflow patterns. Additionally, it should be required that the workflow system can be very tightly integrated with the rest of the enterprise system.

Requirements

Business processes include rules and business logic, for example 'if the value of a purchase order is more than 1000 'then' it needs additional approval. Some business logic should be implemented by the vendor of the platform and should never be changed. For example, when a part is taken from an inventory the stock should always be decreased. Or if a document is locked for editing nobody should be able to change it.

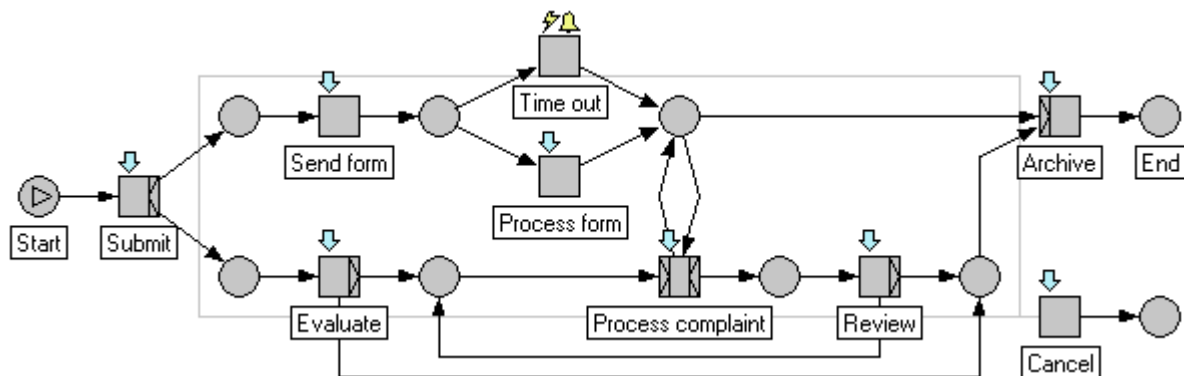


Figure 1. Graphical workflow definition for a request.

Other business processes may be very much dependent on how a specific organization works. For example, an organization may define various processes for dealing with customer requests depending on the type of requests. Based on feedback from customers these processes may be changed over time. It is therefore important that an enterprise system allows users to quickly define and modify processes without the need to request a new software release from the vendor. These are the kind of processes that should be defined with a workflow management system.

Because the business processes need to closely cooperate with the business data, the integration between processes and data needs to be tight. A workflow should be able to control the access rights of the data that it is managing. For instance, when a document goes from state 'draft' to state 'approved', the document access right should go from 'full' to 'view'. To be able to have such behavior combined with good performance, the workflow system needs to be very tightly integrated with the rest of the system.

Since the business processes must be understandable for the users who are participating in the workflow, a good graphical environment to present the workflow is essential. Figure 1 shows an example of a graphical representation for a request workflow. A graphical representation is much easier to understand for users than a program or a script. Therefore, it is desirable to have a lot of expressive power in the graphical representation of the workflow, which leaves nothing or little to be defined in additional scripting or programming.

Graphical visualization is not just helpful for designing a workflow but also for providing feedback to users when the workflow is executed. A common complaint with many workflow systems is that users do not know what happens after they click on a button. Or if a task has been assigned they cannot easily see the current state of the workflow. Figure 2 shows how a graphical environment can help here. If a user receives a task 'Evaluate' the workflow diagram shows the current state in black and previous and next states in grey. The diagram shows that depending on whether the user accepts or rejects the request it can be processed or archived.

Because the workflow management system should be integrated with a web-based enterprise application, it is important that the graphical definition and feedback from the workflow can be done in a web-browser. Drawing and editing diagrams cannot be done in a standard web page and therefore a special browser plug-in or Java applet should be provided. A Java applet is probably preferable because it requires no installation and works on multiple platforms.

Workflow management systems should be able to assign tasks to roles instead of individual people. This avoids that the workflow stalls when a specific person is not available or has left the company. For example, after an employee submits maintenance request a task is created for the role 'facility manager'. Depending on how the workflow is defined one or more persons who have been assigned to that role will receive the task.

Regarding the expressive power of the graphical representations, the workflow management system should be able to

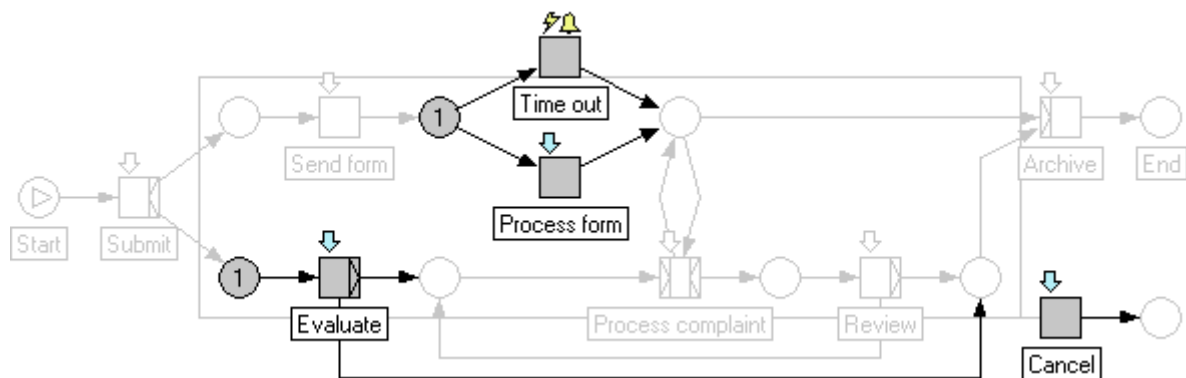


Figure 2. Graphical feedback during execution of a request workflow.

model the workflow patterns as mentioned before. Further on in this paper we will explain the actual patterns in more detail. Finally, it is important that the workflow definition has a formal foundation. This ensures that the behavior of the workflow is well-defined and not dependent on a specific implementation. Furthermore, a formal foundation provides a framework to be able to prove certain properties of a workflow and facilitates integration with analysis tools

Petri Nets

Because of the requirements for a formal foundation and graphical representation, it makes sense to use Petri nets as a basis for workflow definitions [3]. Petri nets were devised in 1962 by Carl Adam Petri as a mathematical tool for modeling and analyzing processes. One of the strengths of this tool is that it enables processes to be defined graphically. Despite the fact that Petri nets are graphical, they have a strong mathematical foundation. Unlike many other schematic techniques, they are entirely formalized.

A straightforward application of Petri nets can model workflow processes to some extent, but there are also some limitations [4]. Therefore, a workflow language YAWL (Yet Another Workflow Language) is proposed in [4] that solves most of these limitations. Axxerion includes concepts from Petri nets and YAWL as a basis for its workflow management module.

Definition

Figure 3 shows the symbols that are used in the Axxerion workflow management module. Generally speaking, a workflow definition consists of places and tasks, connected by connectors.

Each workflow has one start place. When the workflow is executed, a token is put in the start place. Subsequently, the token can enable one or more tasks. If the task is an or-join task, only one token needs to be present in its input places to make it enabled. If the task is an and-join task, all of its input places need to contain a token (there are exceptions however, as we will see later).

An enabled task can be executed, resulting in input tokens being consumed and output tokens being produced. An or-split task

creates only one token, for example a request task produces a token for the state 'rejected' or for the state 'accepted'. An and-split task creates tokens for all enabled outgoing connectors. For example, when making a reservation a token is created for activating a task for reserving a room and another token for reserving equipment.

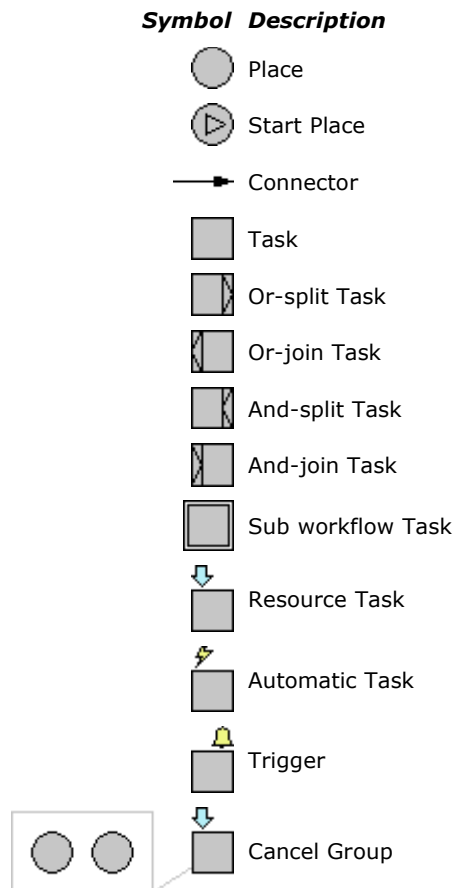


Figure 3. Workflow symbols.

Outgoing connectors from a task can have conditions attached to them. An output token is only generated when these conditions are satisfied. Conditions attached to connectors are based on business data of the underlying object and can be easily specified in the user interface. For more complex conditions, it is possible to use scripts.

It is also possible to attach commands to outgoing connectors. When executing the workflow, such a command can be activated by the user. Execution of a command can result in an output token being created at the end of the connector. Optionally a command can update business data as well, directly or via scripts. In the user interface, commands and conditions

can be displayed as labels to the connectors.

A resource task is a task that is assigned to users from specified user groups. When such a task is enabled during the execution of the workflow, tasks are generated for all the members of these user groups. The users can then pull the task from their task list and execute it. It can be specified in the outgoing connectors how many users should complete the task. For instance, it can be specified that two users from the user group 'Reviewer' need to approve a document before it can be published. An automatic task is a task that is automatically completed by the system during execution of the workflow. One useful application of an automatic task, in combination with a trigger, is to define a 'time-out' task. After a specific time defined by a trigger, the time-out task will be automatically completed. For instance, if the workflow is waiting on a form to be completed by a user, but the user has not reacted for a week, the workflow can automatically advance to the next step by executing the time-out task. In the workflow of figure 1 and 2 an example is shown of such an automatic time-out task.

Workflow Execution Example

We will explain the execution of a workflow in the Axxerion workflow management module according to a relatively simple quotation workflow example (see figure 4).

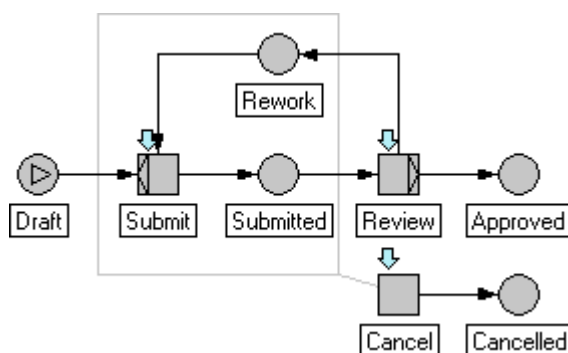


Figure 4. Quotation workflow definition.

The quotation can have 5 states: 'Draft', 'Submitted', 'Rework', 'Approved' and 'Cancelled'. These states correspond to the 5 places in the workflow. There are three tasks: 'Submit', 'Review' and 'Cancel'. The task 'Submit' is an or-join task which is carried out by the creator of the quotation. The 'Review' task is an or-split task,

carried out by the sales manager. The task 'Cancel' is a task connected to a cancel group (depicted by the grey rectangle), and can be carried out by the creator or the sales manager.

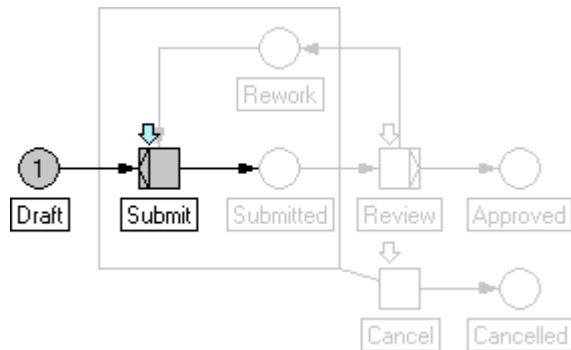


Figure 5. Token in start place, Submit task enabled.

When the workflow is started, one token is put in the start place 'Draft'. Thus, task 'Submit' is enabled and assigned to the creator. This is shown in figure 5. Enabled places and tasks are drawn in black, the rest is drawn in grey. When the submit task is enabled, the user can enter data in the screen displayed in figure 6. Most fields are editable. The buttons at the bottom of the screen correspond to commands that are specified with the outgoing connectors of the tasks that the current user has. In the screen in figure 6, the 'Submit' button is displayed, because this corresponds to the command that is attached to the outgoing connector from the 'Submit' task (the 'Back' and 'Ok' buttons displayed in figure 6 are standard system buttons, that are not related to tasks).

Figure 6. Quotation screen in draft state.

When the task 'Submit' is executed (i.e. the 'Submit' button is pressed), the token from place 'Draft' is consumed and a new token is produced in place 'Submitted' (see figure 6). Now the task 'Review' is enabled and assigned to the sales manager. In addition the task 'Cancel' is enabled because there is a token in the cancel group.

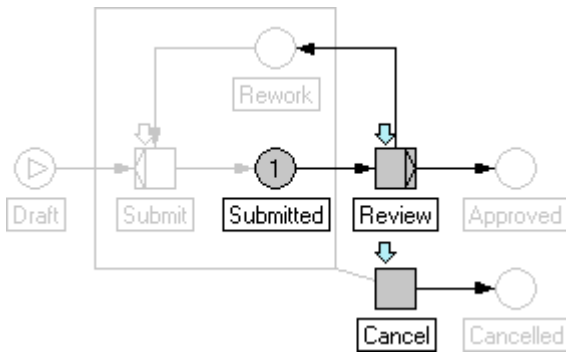


Figure 7. Quotation submitted.

When the sales manager enters the system, the 'Review' task is displayed in the task list. When this task is opened, the quotation screen is displayed, see figure 8. The buttons at the bottom of the screen correspond to the outgoing connectors of the tasks for the sales manager. The 'Cancel' button corresponds to the outgoing connector from the 'Cancel' task, and the 'Rework' and 'Approve' buttons correspond to the outgoing connectors from the 'Review' task. Compared to figure 6, the access rights for the fields have changed. Access rights depend among others on user group and workflow state. In the submitted state, the sales manager can look at the data, but can not modify the data.

Review: Decide if the quotation can be approved.			
ID	QUI-040039	Type	Incoming quotation
External reference	2004-0345	Quotation date	10-05-04
Belongs to	Quotation request	Due date	18-05-04 10:50
From	Gemeentebedrijf	Start	08-06-04
To	Installatiebedrijf	End	15-06-04
Name	Quotation request	Labour	3.55 Hour
Description	Please send a quotation for the specified items.	Amount	103.65 EUR
		Vat	0.00 EUR
Contact	Muys, Ronald - Installatiebedrijf	Total	103.65 EUR
Status	Submitted		
<input type="button" value="Cancel"/> <input type="button" value="Rework"/> <input type="button" value="Approve"/> <input type="button" value="Back"/> <input type="button" value="Ok"/>			

Figure 8. Quotation screen in submitted state.

In the task 'Review', the sales manager can decide if the quotation can be approved or if it needs more work. If it is decided to rework the quotation (the 'Rework' button is pressed), the token will go to the place 'Rework' and the 'Submit' task will be enabled again (see figure 9). Note that the task 'Cancel' is still enabled because there still is a token in the cancel group rectangle.

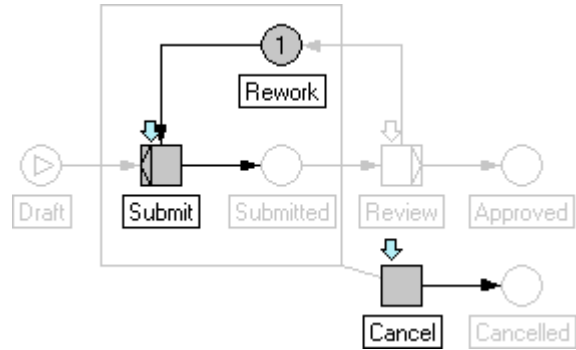


Figure 9. Quotation needs more work.

Now the quotation can be submitted again, and after another review it can be approved (see figure 10). When the token is in the place 'Approved', there are no more enabled tasks. Completion of the workflow is implicit: if no more tasks are enabled, the workflow is completed.

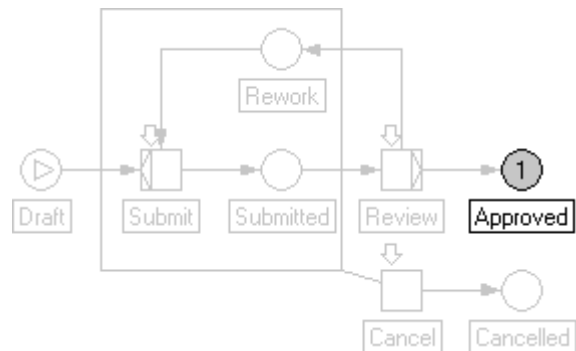


Figure 10. Quotation approved, workflow completed.

This quotation workflow is a simple example. The workflow in figure 1 and 2 is more complicated because there are parallel paths and there is a synchronization point.

Implementation

The Axxerion workflow management module is implemented in Java in a J2EE environment. The workflows can be designed and executed in a web browser, using html and a Java applet. This enables dynamic placing and moving of the various workflow items. The pictures in this paper are all generated from this applet, so this is also what users can see. Because the workflow system is very tightly integrated with the rest of the Axxerion workplace automation system, performance of the application can be optimized to a large extent. This is very important, because the workflow is used for access rights checking as well. To render a screen

in this system, potentially very many access rights checks have to be made. All data required for the workflow definition is stored in a relational database and can be changed dynamically without recompiling the system. The applet for workflow design and visualization is downloaded automatically, which typically takes less than 2 seconds. All communication between the applet and the server is handled via XML over HTTP(s) so that no special network infrastructure is required. On the server side, the workflow definitions can be exported to XML and imported on other servers running Axxerion, regardless of the underlying relational database.

Workflow Patterns

Different perspectives can be taken to evaluate a workflow system. The control flow perspective is the main perspective and deals with the workflow structure, the tasks and their execution. Other perspectives such as the data or organizational perspectives rest on it. The workflow patterns [2] mainly consider the control flow perspective of the workflow system. To assess the capabilities of the Axxerion workflow management module we will evaluate it according to these workflow patterns. The patterns range from fairly simple constructs to complex routing primitives.

Pattern 1: Sequence

In the sequence pattern, a task B in the workflow is enabled after completion of another task A in the same process. This pattern can be modelled in Axxerion as shown in figure 11.

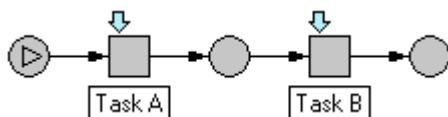


Figure 11. Sequence.

This pattern can also be seen in the quotation example we showed: Task 'Review' is enabled after task 'Submit' has been completed.

Pattern 2: Parallel Split

The parallel split pattern is defined as a point in the workflow where a single thread of control splits into multiple threads of

control which can be executed in parallel, thus allowing tasks to be executed simultaneously or in any order. This pattern is supported in Axxerion by the and-split task, as shown in figure 12. Task A and B are both executed, in any order.

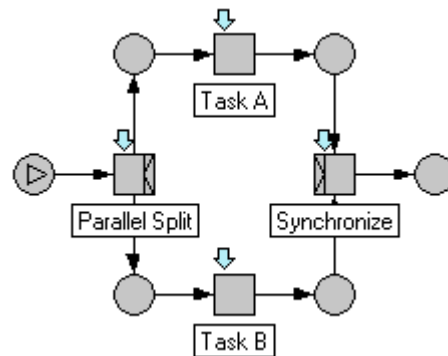


Figure 12. Parallel split and synchronization.

The parallel split can also be seen in figure 1. The initial single thread for the request splits in two threads: one to send a form, and one to evaluate the request.

Pattern 3: Synchronization

The synchronization pattern is a point in the workflow where multiple parallel threads converge into one single thread of control, thus synchronizing multiple threads.

This pattern is supported in Axxerion by the and-join task. An example can again be seen in figure 12 and figure 1. At the end of the workflow in figure 1, the 'Archive' task is executed, which synchronizes the two input threads and produces one output token.

Pattern 4: Exclusive Choice

The exclusive choice pattern is defined as a point in the workflow where, based on a decision, one of several branches is chosen.

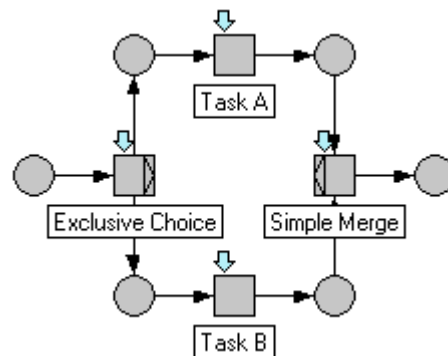


Figure 13. Exclusive choice and simple merge.

This pattern is supported in Axxerion by the or-split task, as shown in figure 13. Either Task A or Task B will be executed depending on the choice that the user makes.

Another example of this pattern can be found in figure 4 in the 'Review' task. Depending on the sales manager's decision, the 'Approved' or the 'Rework' branch is chosen.

Pattern 5: Simple Merge

The simple merge pattern is defined as a point in the workflow where two or more alternative branches come together without synchronization. It is an assumption of this pattern that the incoming branches are not executed in parallel.

This pattern is supported in Axxerion by the or-join task, see figure 13.

Branching and Synchronization

The following patterns are more advanced patterns for branching and synchronization. As opposed to the previous patterns, these patterns do not have straightforward support in most workflow engines.

Pattern 6: Multi Choice

The multi choice pattern is defined as a point in the workflow where, based on decisions, a number of branches is chosen. In Axxerion, this pattern is implemented as an and-split task with conditions on the connectors. The user can select in the form belonging to the workflow which branches should be taken. In figure 14, an example is shown. When the user carries out the task 'Submit', any combination of a flight, hotel and car booking can be selected. When 'Submit' is executed, the workflow engine can determine the branches that should be taken by examining the

conditions on the outgoing connectors. Suppose that a flight and a hotel booking have been selected, then the workflow execution looks like figure 15.

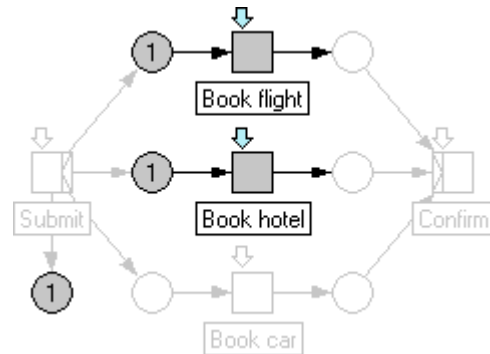


Figure 15. Multi choice, taking two branches.

The book flight and book hotel branches have been chosen, and there is no token in the book car branch.

Pattern 7: Synchronizing Merge

The synchronizing merge is defined as a point in the workflow where multiple paths converge into one single thread. If more than one path is taken, synchronization of the active threads needs to take place. If only one path is taken, the alternative branches should reconverge without synchronization.

The synchronizing merge is implemented in Axxerion with the and-join task. An and-join task can optionally be linked to a corresponding and-split task. At runtime, the and-join task can query the corresponding and-split task that was executed about the branches that were taken. Using this information, it is possible that the and-join task is enabled without all its input places containing tokens. Instead it can be enabled by just the branches that were taken. It is possible that the whole

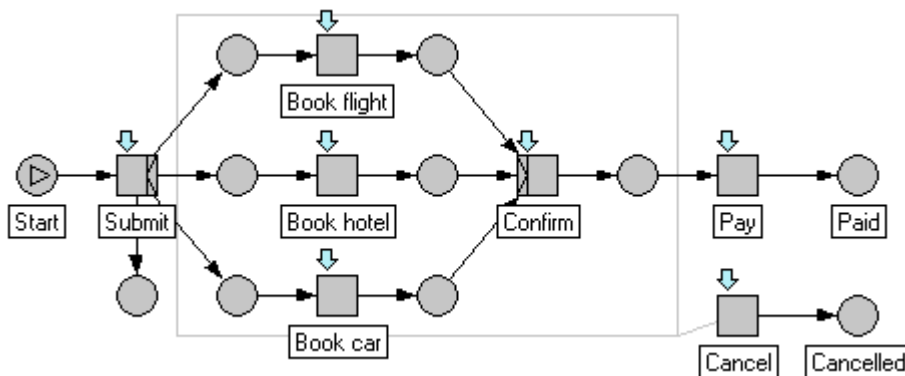


Figure 14. Travel request workflow, showing multi choice and synchronizing merge.

“and-split-join-block” is carried out inside a loop.

Note that in this approach it is not allowed for tokens to “escape” or “enter” the paths in between the and-split and the and-join. The same number of tokens that leaves the and-split task should arrive at the and-join. For instance, it is not allowed that the 'Book flight' task is an or-join task that takes tokens from another execution path.

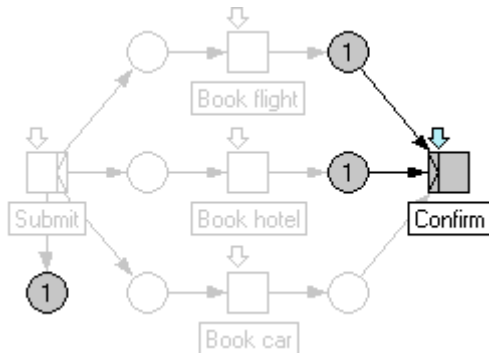


Figure 16. Synchronizing merge, enabled by 2 tokens.

An example is shown in figure 16. The flight and hotel have been booked, and the 'Confirm' task (which is a synchronizing merge) is enabled.

Pattern 8: Multi Merge

The multi merge is defined as a point in the workflow where two or more branches reconverge without synchronization. If more than one branch gets activated, possibly concurrently, the task following the merge is started for every activation of every incoming branch.

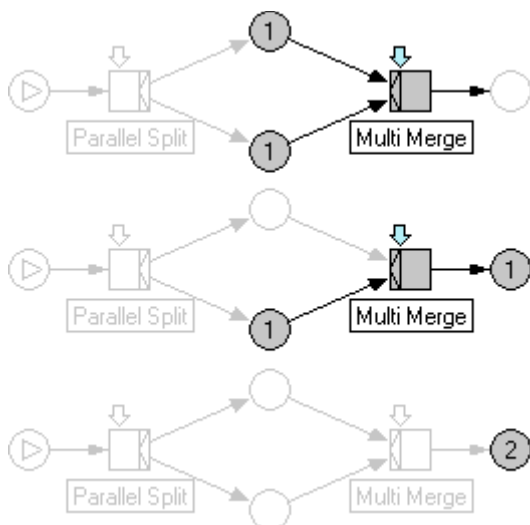


Figure 17. Multi merge execution.

This pattern is implemented in Axxerion using the or-join task. In figure 17 the effect of this pattern is visualized. At first there are two tokens active in parallel. When the multi merge task is executed, one token is put in the output place. Then the multi merge task is still enabled, because it consumed only one token. When it is again executed, a second token is put in the output place and the multi merge is finished.

Pattern 9: Discriminator

The discriminator is a point in the workflow that waits for one of the incoming branches to complete before activating the subsequent task. It then ignores all remaining branches.

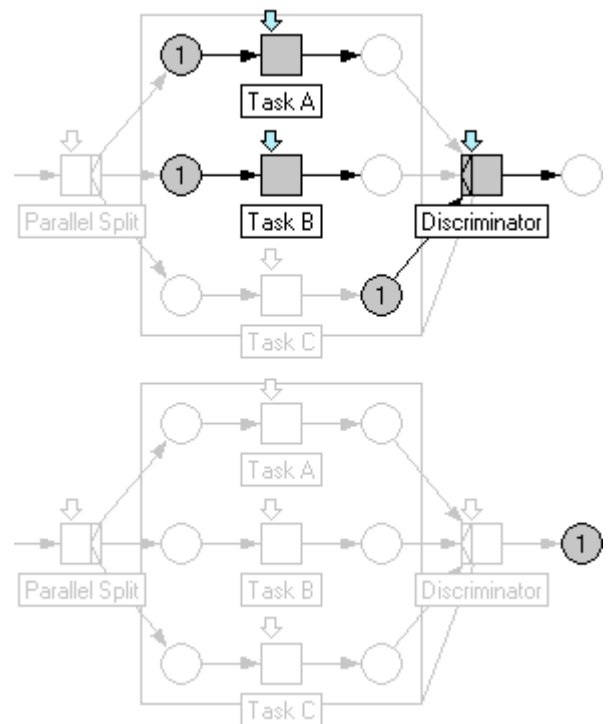


Figure 18. Discriminator execution.

The discriminator can be implemented in Axxerion using the or-join task in combination with a cancel group, see figure 18. The discriminator task is enabled when either task A, B or C has been executed. When the discriminator is executed, all tokens in the cancel group are 'cleaned up'.

Pattern 10: Arbitrary Cycles

An arbitrary cycle is defined as a point in the workflow where one or more tasks can be done repeatedly. An example of how this pattern is supported in Axxerion is shown in figure 19.

The parallel split task splits into a branch to create a delivery, and a branch to decide if another delivery is needed. If yes, the parallel split is executed again, thus creating another delivery task.

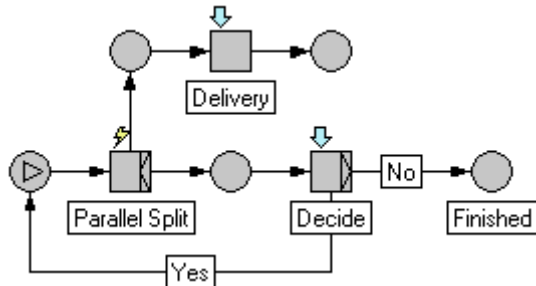


Figure 19. Arbitrary cycles.

Pattern 11: Implicit Termination

A given subprocess should be terminated when there is nothing else to be done. In other words, there are no tasks in the workflow and no other tasks can be made active.

This is standard behavior in the Axxerion workflow management module. For instance, in figure 17 the workflow has ended when all tokens are in the final place (the place that has no outgoing connectors).

Multiple Instances

The following patterns involve tasks that have multiple instances running during execution of the workflow. There are two types of requirements. The first requirement is the ability to launch multiple instances of a task or subprocess. The second requirement is the ability to synchronize these instances and continue after all instances have been handled.

Pattern 12: Multiple Instances Without Synchronization

Within the context of a single workflow instance, multiple instances of a task can be created. To see how this pattern can be implemented, we can again look at figure 19. During execution of this workflow, multiple instances of the task delivery can be created.

Actually, multiple task instance support is built-in in Axxerion at a lower level as well. For any task, user groups can be specified that should carry out the task. When the task is instantiated during the course of the workflow, tasks are created for each person

in the user group. These persons see the task in their task list, and can then decide to handle the task. In addition, with every outgoing connector from a task, it can be specified how many task instances should be executed to advance on the branch.

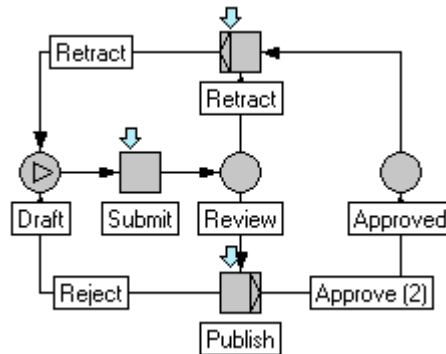


Figure 20. Document approval workflow.

For instance, in the document approval workflow of figure 20, the task 'Publish' is assigned to the user group 'Reviewer'. To go from state 'Review' to 'Approved', two users from the reviewer user group need to approve the document. If only one of them rejects the document, it is put back in the state 'Draft'.

Pattern 13: Multiple Instances With a Prior Design Time Knowledge

For one process instance a task is enabled multiple times. The number of instances of a given task for a given process is known at design time. Once all instances are completed some other task needs to be started.

This pattern can be implemented in Axxerion in various ways. A first example is the document approval workflow in figure 20. At design time, it can be specified that all reviewers should get the 'Publish' task. It can also be specified how many reviewers should approve before the document is approved.

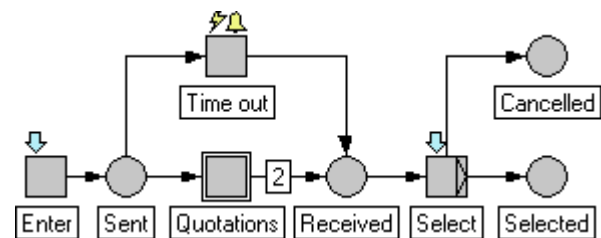


Figure 21. Request for quotation workflow.

Another example is displayed in figure 21. This is a request for quotation workflow. It

uses sub workflows for the actual quotations. When the 'Quotations' sub workflow task is enabled, instances of the quotation workflow (see figure 4) are executed that are linked with the main request workflow. From a data perspective, the quotation objects are linked to the request object, and the workflow instances are linked to the quotations and request. When the request for quotation workflow is started, first general information on the request is entered, like a description, needed items, the due date, etc. The request is then sent out to a number of contractors, who will have to send back a quotation before the due date. This state of the workflow is displayed in figure 22.

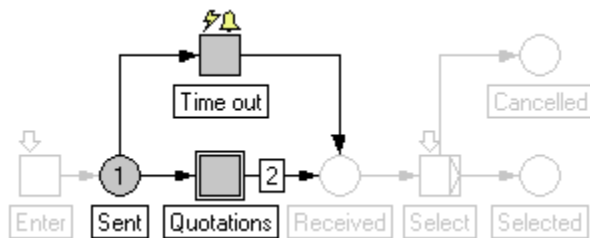


Figure 22. Main workflow is waiting for quotations.

The system automatically creates a specified number of incoming quotations (in this case 2), that are ready to be filled out when the quotations from the contractors come in. When more quotations come in, they can be added at runtime, and they will be linked to the main request workflow as well. When the incoming quotations have been approved or cancelled, they signal the main workflow of the completion of a sub workflow.

In the workflow definition, it has been specified that the connector going from the 'Quotations' subworkflow task to the 'Received' place is enabled when 2 incoming quotations are approved. So when 2 quotations have been approved, the main workflow will advance to the 'Received' state.

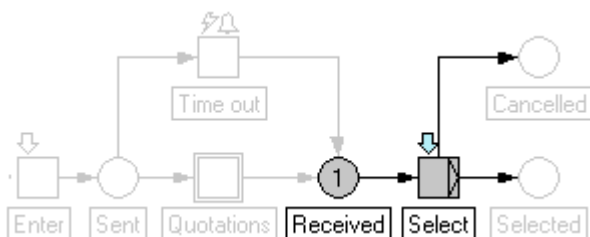


Figure 23. Sub workflows have finished.

Subsequently, one quotation can be selected by the purchase manager in the 'Select' task. In this task, the purchase manager can also decide to cancel the request, see figure 23.

If there are no 2 approved quotations before the due date, the automatic time-out task will be executed. This will result in all non-completed quotations to be cancelled, because the sub workflows are not enabled anymore. The purchase manager can then still decide what to do with the completed and/or cancelled quotations.

Pattern 14: Multiple Instances With a Priori Runtime Knowledge

For one case a task is enabled multiple times. The number of instances of a given task for a given case varies, but is known at some stage during runtime before the instances of the task have to be created.

Once all instances are completed some other task needs to be started. This pattern is supported by Axserion as shown in figure 21. The number of quotations can be specified at runtime.

Pattern 15: Multiple Instances Without a Priori Runtime Knowledge

For one case a task is enabled multiple times. The number of instances of a given task for a given case is not known during design time, nor is it known at any stage during runtime before the instances of the task have to be created. Once all instances are completed some other task needs to be started. The difference with pattern 14 is that even while some of the instances are being executed or already completed, new ones can be created.

This pattern is supported as well by Axserion, as shown in figure 21. At runtime, quotations can be added that automatically link as a sub workflow to the main request workflow.

State-based Patterns

In real workflows, most workflow instances are in a state awaiting processing rather than being processed. In many workflow systems the notion of state is interpreted in a narrower fashion and is essentially reduced to the concept of data. As this section will illustrate, there are scenarios where an explicit notion of state is required.

Pattern 16: Deferred Choice

The deferred choice pattern is defined as a point in the workflow process where one of several branches is chosen. In contrast to pattern 4 (exclusive choice), the choice is not made explicitly, but several choices are offered to the environment. Once a branch is activated, the other alternative branches are withdrawn. It is important to note that the moment of choice is delayed until processing in one of the alternative branches is actually started. The moment of choice is as late as possible.

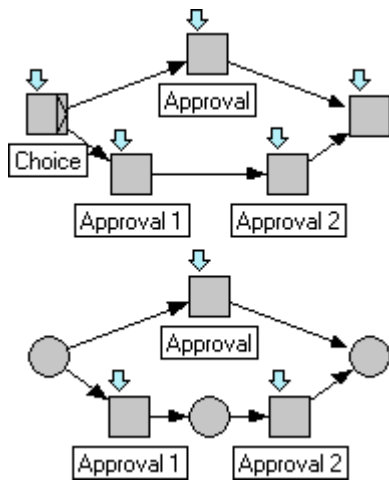


Figure 24. Difference explicit and implicit choice.

Figure 24 illustrates the difficulty in supporting this pattern when the workflow language does not support states directly. The top workflow is made using only tasks, the bottom workflow also has states (the circles). The workflow is about a document approval. The approval can be made by the manager, or alternatively by two coordinators. In the top workflow, the only way to make the distinction between the branches is make a separate task to decide which approval to take. In the bottom workflow, this decision is made implicitly by executing an approval in one branch. Since Axxerion has the notion of states, this pattern is straightforward to implement. Another example can be seen in figure 21. The choice between time-out or advancing when 2 quotations are approved is only made at the time one of these steps is completed. Such a time-out is difficult to model when the workflow language does not support states.

Pattern 17: Interleaved Parallel Routing

A set of tasks is executed in an arbitrary order. Each task in the set is executed, the order is decided at runtime, and no two tasks are executed at the same moment. This pattern can be implemented directly in Axxerion because states are supported.

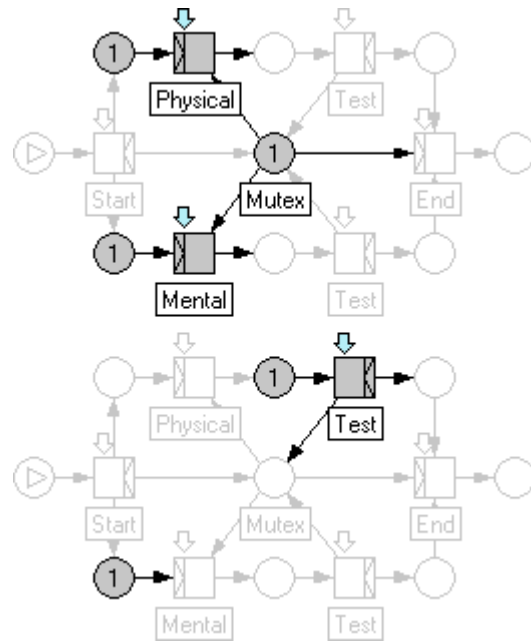


Figure 25. Interleaved parallel routing.

In figure 25, an example of interleaved parallel routing is shown. In this example, the Navy requires every job applicant to take two tests: a physical test and a mental test. These tests can be conducted in any order but not at the same time. To realize the desired behavior, a mutual exclusion place is added to synchronize tokens. As can be seen in figure 25, initially both tests are enabled. After the physical test has started, the mental test is not enabled. When the physical test has ended, the mental test will be enabled again.

Pattern 18: Milestone

The enabling of a task depends on the case being in a specified state. The task is only enabled when a certain milestone has been reached, that is in another parallel thread of execution.

The milestone pattern can be implemented directly in Axxerion. An example is shown in figure 26. This example is a step further in the execution than what is shown in figure 2. The milestone is the state where the two parallel paths are synchronized.

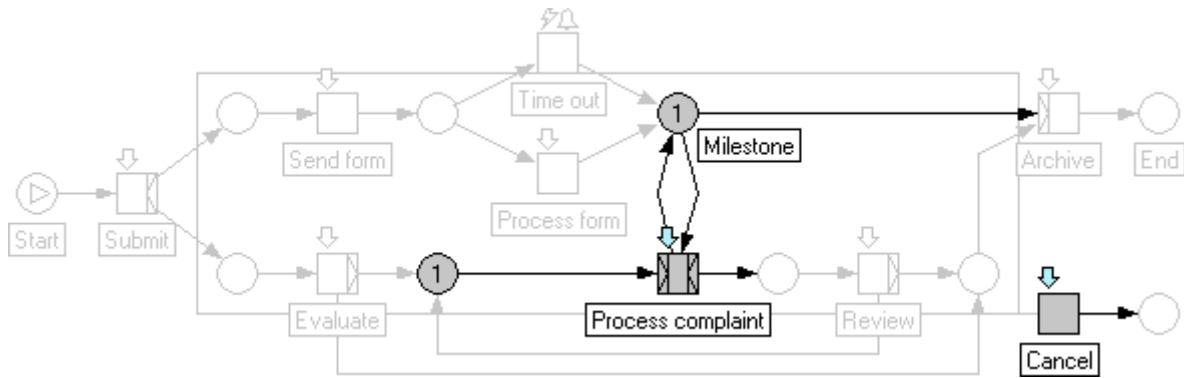


Figure 26. Milestone.

Cancellation Patterns

The following patterns deal with the ability to cancel tasks in a workflow.

Pattern 19: Cancel Activity

An enabled task is cancelled, meaning that the thread waiting for the execution of the task is removed. The cancel activity or task pattern can be implemented directly in Axxerion. We have seen several examples of this pattern related to the deferred choice pattern. For instance, in figure 2, when the time out task is executed, the process form task is cancelled.

Pattern 20: Cancel Case

A workflow instance or case is removed completely. Even if parts of the workflow are instantiated multiple times, all descendants are removed. For this pattern, the cancel group can be used. When a task connected to a cancel group is executed, all tokens in the cancel group are removed. Also, all tasks related to those tokens are cancelled. For instance in figure 26, the workflow case can be cancelled at once by executing the cancel task.

Scripting Support

All patterns described in this paper have been created graphically in a web browser without the need for any programming. Axxerion does however offer scripting for defining more complex decision rules and business logic. The scripts are written in Java and are compiled at run time. Scripts can be used as conditions on connectors, for example an approval branch is only enabled when the 'amount' data field in an invoice exceeds '1000'. Scripts can also be used to send messages to other systems, call on internal APIs in the

Axxerion system or to create, delete or update information. Scripting does require a good knowledge of programming and data structures and is typically only used by advanced users and system integrators. A detailed discussion on the Axxerion workflow scripting capabilities is beyond the scope of this paper.

Conclusion

We have given an overview of the workflow functionality offered in the Axxerion workplace automation system. The system was evaluated according to the workflow patterns in [2]. It has been shown that Axxerion supports all workflow patterns in a direct way, without the need to resort to additional scripting or programming. It has also been demonstrated how advanced workflow capabilities can be integrated in a graphical web environment. The focus of this paper has been mainly on the control flow perspective of the Axxerion system. For an overview of the other aspects of the system we refer to the Axxerion whitepaper [5].

Authors

Martin Waardenburg and Maarten van Emmerik each have over 15 years experience in development of enterprise systems for computer-aided design and business process management. They founded Axxerion to develop a new generation workplace and facility automation software. Workflow management, based on the concepts in this paper, is a key element in this software.

References

[1] Workflow Management Coalition, non-profit organization (www.wfmc.org).

[2] Workflow Patterns - W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski and A.P. Barros, 2002 (www.workflowpatterns.com)

[3] Workflow Management, Models, Methods and Systems – Wil van der Aalst and Kees van Hee, MIT Press, 2002

[4] YAWL: Yet Another Workflow Language – W.M.P. van der Aalst and A.H.M. ter Hofstede, 2003 (www.citi.qut.edu.au/yawl)

[5] Workplace and Facility Automation via the Internet – Maarten van Emmerik and Martin Waardenburg, 2004 (www.axxerion.com)