# Workflow patterns

COSA Workflow software


| | |
|---|---|
| **Reference** | **Workflow Patterns** |
| **Version** | **1.0 – Released version** |
| **Author** | **TRANSFLOW Nederland BV** |
| **Date** | **February 18, 2003** |

## Management summary

TRANSFLOW (formerly Software Ley, COSA Solutions) has been delivering high volume production Workflow products for more than two decades. We have gained an excellent reputation implementing complete and comprehensive Workflow solutions based on our COSA products.

The quality of a Workflow product and therefore the ultimate success of an implementation is determined by several essential factors. One of those factors is the ability to model real life business processes in both an efficient and practical manner. The modeling quality of a Workflow product has a profound impact on the flexibility, performance and maintainability of a Workflow system and hence it's overall total cost of ownership (TCO).

TRANSFLOW has recognised the importance of modeling and specifically chose to use the scientifically based Petri Net methodology for modeling, analysing and running its Workflow processes. The mathematical and logical principles of Petri Net models not only enhance the above, they also guarantee the consistent and faultless processing of cases.

In order to assess the modeling requirements for Workflow, the Technical University of Eindhoven in the Netherlands and the Technical University of Queensland in Australia conducted a comprehensive, scientific research project in 2000. Based on empirical research, 20 required process patterns were defined. These patterns formed the basis for an in-depth evaluation and comparison of the leading Workflow Management products.

The analysis of COSA 3.0 was carried out in co-operation with TRANSFLOW Nederland and we endorse the conclusions, including those concerning our own product. Since the research took place, the COSA Workflow product has undergone several enhancements, whilst new functional concepts such as Case Management are/have been introduced. The essence of this paper is to explain the significance of the different patterns and to describe not only that we support them but, more importantly, *how* we support them. Our information is primarily based on the latest versions 4.0/4.1 of COSA Workflow Management. Some features of COSA Workflow 4.2, which will be released at the beginning of 2003, are also mentioned

## Patterns support by COSA Workflow

| Pattern Number | Process pattern | Support by COSA Workflow Management |
|---|---|---|
| 1 | Sequence | + |
| 2 | Parallel Split | + |
| 3 | Synchronization | + |
| 4 | Exclusive Choice | + |
| 5 | Simple Merge | + |
| 6 | Multi Choice | + |
| 7 | Synchronization Merge | +* |
| 8 | Multi Merge | +* (+) |
| 9 | Discriminator | +* |
| 10 | Arbitrary Cycles | + |
| 11 | Implicit termination | +* (+) |
| 12 | Multiple Instance without synchronization | + |
| 13 | Multiple Instance with prior design knowledge | + |
| 14 | Multiple Instance with prior run-time knowledge | + |
| 15 | Multiple Instance without prior run-time knowledge | +* |
| 16 | Deferred choice | + |
| 17 | Interleaved parallel Routing | + |
| 18 | Milestone | + |
| 19 | Cancel Activity | + |
| 20 | Cancel Case | +* |

+ Supported
+* Supported with the use of other COSA WfM functions
(+) Support level in COSA 4.2 release Q1 2003

## Introduction

Since the introduction of the COSA Workflow product suite in 1990, COSA has gained an excellent reputation when it comes to the implementation of high volume production workflow. Our customers have for many years and to their complete satisfaction, run solutions based on our technology. Whilst we consider our pro-active role during implementation as a key success factor, we also regard the quality of several different aspects of a product to be extremely relevant. In our view, the quality of any business process automation solution is determined largely by implementation services but at the same time, it is also restricted by the limitations of the underlying technology.

We consider four aspects to be essential for a Workflow product:

- The expressive power of the model constructions
- Possibilities of organisation modeling and the dispatching of work
- The ease with which the product can be integrated with external business applications
- General IT requirements and adherence to industry standards

In this paper we will specifically concentrate on one of the four aspects that determine the quality of Workflow, the expressive power of the process constructions modeling. This expressive power or modeling quality, has a profound impact on the flexibility, performance and maintainability of a Workflow implementation and hence it's overall total cost of ownership (TCO).

This paper is based on the original scientific research into Workflow process patterns, carried out by the Eindhoven University of Technology and the Queensland University of Technology. The research project was led by prof. W. van der Aalst and an abstract can be downloaded from **(http://tmitwww.tm.tue.nl/research/patterns/).**
Mr. Van der Aalst's research proves without doubt that there are a number of clearly defined patterns or process constructions, which are absolutely essential for automating any business process. Different Workflow products not only tend to support these patterns in different ways, many Workflow products simply do not support all of the required modeling constructions. To validate this assumption, Van der Aalst made an in-depth evaluation and comparison of the leading Workflow products available on the market today.

We are convinced that the process modeling possibilities, flexibility and ways in which work can be dispatched within an organisation, ultimately determines the manageability and the maintainability of a solution. In short, every 'work around' required has a direct and profound (often: negative) influence on the complexity and clarity of a solution, seen in the academic light of Petri net modeling.

We endorse the conclusions drawn by Mr. Van der Aalst, including those concerning our own product. His analysis of COSA Workflow 3.0 was carried out in co-operation with TRANSFLOW Nederland. We would however like to add that since the evaluation of COSA (in early 2000) we have released COSA Workflow 3.1 and COSA Workflow version 4.0 / 4.1 both offering many new enhancements, also regarding the modeling aspects. Next to that, we have introduced a Case Handling concept (the COSA Activity Manager) in 2001, which was obviously not analysed in the original research.

In this paper, all the patterns described in Mr. Van der Aalst's research will be reviewed. We will not only show that we support each individual pattern, but more importantly, how we support it. We have also taken into account the latest information, with regards to the modeling facilities of our recently released COSA Workflow version 4.0 / 4.1.

## The underlying Modeling Methodology

In contrast to many other Workflow systems, the COSA WfM uses the Petri Net theory for modeling, analyzing and running its processes. After almost 40 years and more than 10.000 scientific publications it has gradually been proved without any doubt that the mathematical and logical principles of Petri Net models guarantee an optimum, consistent and faultless processing of cases. The use of Petri Nets is a formalism that imposes an exact description, thus preventing ambiguities, obscurities and contradictions in process models. Furthermore, the Petri Nets of COSA are so-called High-Level and Time-Coloured Petri Nets.

Petri Net models do not only contain the process-relevant activities but also the states before and after an activity. The addition of states in the process models provides logical consistency and increases the modeling precision and the possibilities to add conditions for process control. Petri Nets make it possible to create recurrent, essential process constructions and algorithms that could not be easily supported by other Workflow packages without all kinds of workarounds. To put it bluntly, Petri-nets help to keep even highly structured and complex processes simple. In practice, this means that COSA WfM is an excellent system, for example, for processes and activities that must be executed automatically without end user intervention. This kind of workflow is also known as “straight through processing (STP)” or “unattended operation”. Examples are the automatic starting or stopping of batch operations, the automatic control of business applications and the control of e-business processes and other Enterprise Application Integration (EAI) implementations.

**Parallel and conditional processes**

The most important advantage of Workflow Management lies in the fact that it enables processes and activities to be conducted and coordinated parallel rather than sequential. Economically speaking, Workflow improves the productivity of an organization because different work items can be carried out simultaneously, by different kind of resources (human, machine). This allows for a more extensive division and specialization of labor within an organization.
In order to run processes in parallel, COSA can call or start (sub)processes. Within COSA WfM, there are two different types of sub-processes. The distinction is made between “synchronous processes” (the main process stops and continues when the sub-process has been completed) and “asynchronous processes” (the main process continues independently from the sub-process). In addition, it is also possible to invoke another process by means of the trigger concept. In this case, specific variables are exchanged with the process to be started. This means that any type of application or event can automatically start a process in COSA

**Time Management**

Petri Nets enable the COSA WfM to provide a level of time management and listing of agenda items for users. These are essential functionalities for monitoring cases, performing automatic time monitoring, planning the personnel and taking care of lead times. Functions as sending reminders for ending deadlines or the automatic startup of escalation procedures are practical examples of Time or Escalation Management with COSA Workflow.

**Simulation and analysis**

An important aspect of Workflow is the possibility to analyze the runtime functionality of their correctness and performance already during the modeling phase. In other words, a process must be correct, must not contain logical errors, must finish in the event of termination and should not leave behind any work. Due to the mathematical basis

of Petri Nets, COSA can dynamically test the modeling rules and automatically provide a validated design. The COSA software has a simulation module in which the process model can be analyzed regarding logical correctness and performance of the process. For thorough and in-depth analyses and simulations, COSA flows can be directly read in othertools such as Exspect (Deloitte & Touche) or Woflan (freeware from the TU Eindhoven). When relevant, real life empirical data from the COSA database is used. This so-called bi-directional link with such tools is unique for COSA WfM.

Simulation can also help to predict the effect of possible future developments on the organization and the quality levels that can be achieved by the company. For example, the effect of a strong growth of customers (or customer related cases) can be simulated and analyzed in respect to durations and number of customers (customer cases) that can be served with the current organization and/or organization structure. Possible changes in the organizational structure can be tested first for their effects before implementation. This highly reduces the risks for companies and makes it possible to have a flexible and optimized organization, ready to adapt to any changes if necessary.

## Workflow patterns

In this section we describe and comment the 20 Workflow process patterns defined by Prof.dr.ir. W. van der Aalst in his scientific research project. These patterns form the fundamentals for successfully implementing Workflow Management systems that require flexibility, consistency and maintainability of business processes. TRANSFLOW uses these and other patterns, together with the Petri net methodology for it's Workflow products and Workflow implementations. As we will show, all our solutions are based on sound scientific and mathematical principles, whilst simplicity and ease of use are retained. Although TRANSFLOW with COSA has propagated high level modeling techniques for well over two decades, only recently, within the last two years, the market has become aware of the importance of the modeling capabilities that individual Workflow products have, or should have to offer. As Mr. Van der Aalst has shown in his research, there are enormous differences and discrepancies when it comes to the modeling strengths of these Workflow products..

Each workflow pattern consists of the following:

- The name of the workflow pattern
- A description of the required functionality
- An example of how the pattern can be used in real life situations
- The solution(s) which can be used to realise the necessary functionalities within the context of COSA workflow
- The importance of the pattern.

We complete the pattern descriptions with an example of a practical business case, created by Mr. Van der Aalst and show how this case is modelled with COSA Workflow.

## Pattern 1: Sequence

***Description***

An activity in a workflow process is enabled after the completion of another (preceding) activity in the same process. Synonyms: sequential routing, serial routing.

***Examples***

- An insurance claim is evaluated after the client's file is retrieved.
- Activity *Activity* is executed after the execution of the previous activity Pattern_1.

The sequence pattern is used to model consecutive steps in a workflow process and should be supported by all of the workflow management systems available. A typical implementation involves linking two activities with an unconditional control flow arrow.


Fig. 1 Sequence

***Support***

COSA WfM supports the sequence pattern.

***Importance of the pattern***

The sequence pattern is a basic control flow pattern and the most easy pattern to be used in a model. Activities are executed in succession and are therefore dealt with in a correct order.

## Pattern 2: Parallel split

***Description***

A point in the workflow process where a single thread of control splits into multiple threads of control which can be executed in parallel, thus allowing activities to be carried out simultaneously or in any order. Synonyms of the parallel split: AND-split, parallel routing, fork.

***Examples***

- The execution of the activity *payment* enables the execution of the activities *ship_goods* and *inform_customer.*
- After registering an insurance claim, two parallel sub-processes are triggered: one for checking the policy of the customer and one for assessing the actual damage.

More than one activity will start at the same time, after one single previous activity has ended.

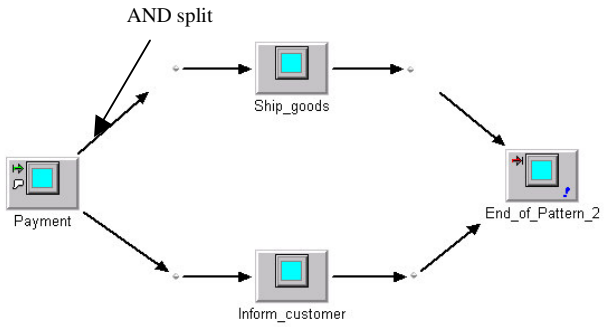



Fig..2 Parallel split

***Support***

The parallel pattern (AND-split) is supported by COSA.

***Importance of the pattern***

The parallel pattern (also a basic control pattern) makes it possible to shorten throughput times, by modeling a split in the process, through which one can use several chosen flows in parallel.

## Pattern 3: Synchronization

***Description***

A point in the workflow process where multiple parallel sub-processes/activities converge into one single thread of control, thus synchronizing multiple threads. This pattern assumes that each incoming branch of a synchronizer is executed only once. Synonyms: AND-join, rendezvous, synchronizer.

***Examples***

- The Activity *archive* is enabled after the completion of both activity *send_tickets* and activity *receive_payment.*
- Insurance claims are evaluated after the policy has been checked and the actual damage has been assessed.

Only after former determined or chosen activities are completed, a flow is ended.

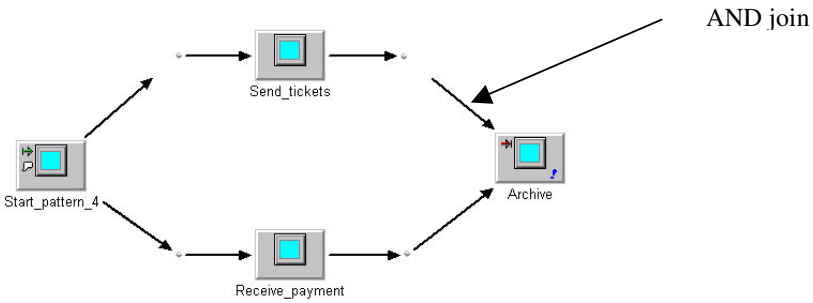



Fig. 3 synchronisation

***Support***

Synchronization of multiple threads is supported by COSA.

***Importance of the pattern***

The synchronization pattern makes sure that all flows from the parallel split are converged correctly. Only then will the process continue.

## Pattern 4: Exclusive choice

***Description***

A point in the workflow process where, based on a decision or workflow control data, one of several branches is chosen. Synonyms: the XOR split, conditional routing, switch, decision.

***Examples***

- The activity *evaluate_claim* is followed by either *pay_damage* or *contact_customer.*
- Based on the workload, a tax declaration is either checked using a shortened administrative procedure or is thoroughly evaluated by senior employees.

The user can make a choice between two activities, but a choice can also be made between two users, for instance based on a variable in the process (if a zip code is >= 4000, then user 1; if < 4000, then user 2).

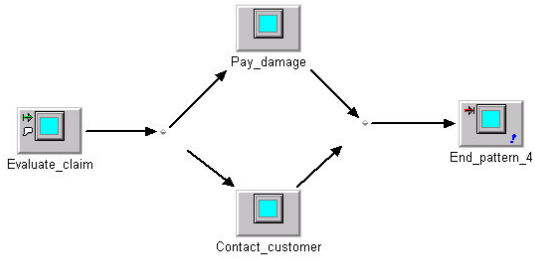



Fig. 4 Exclusive choice

***Support***

The exclusive choice pattern is supported by COSA.

Special conditions can determine which activity is chosen. This possibility anticipates on pattern 6: the multiple choice pattern.

***Importance of the pattern***

The exclusive choice pattern makes it possible that certain paths are chosen automatically within a process, based on certain choices or conditions.

## Pattern 5: Simple merge

***Description***

A point in the workflow process where two or more alternative branches come together without synchronization. It is an assumption of this pattern that none of the alternative branches is ever executed in parallel. Synonyms: XOR-join, asynchronous join, merge.

***Examples***

- The activity *archive_claim* is enabled after either *pay_damage* or *contact_customer* is executed.
- After either the payment is received or the credit is granted, the car is delivered to the customer.

The activity after the merge will only be reached if a single flow has been completed either through Activity 1 (as shown at the top of the illustration) or though Activity 2, as determined during the exclusive choice.

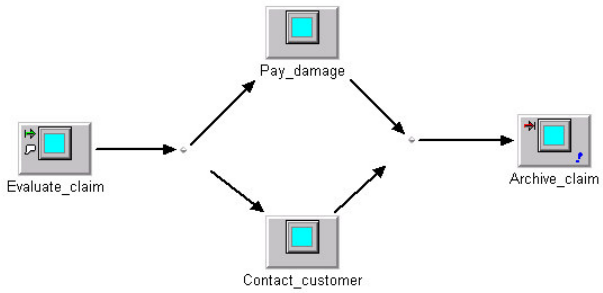



Fig. 5 Simple Merge

***Support***

The simple merge pattern is supported by COSA.

***Importance of the pattern***

To end the split in a process correctly, the simple merge pattern can be used. The process will be able to continue without being restricted by activities that are dependent on choices.

## Pattern 6: Multi choice

***Description***

A point in the workflow process where, based on a decision or on workflow control data, a number of branches are chosen. Synonyms: Conditional routing, selection, OR-split.

***Example***

After performing the activity Choice, the activity Activity_1 *and*/or the activity Activity_3 is executed.. At lease one of these activities is executed but it is also possible that both need to be executed. Activity_2 is always executed in the example.

After carrying out one activity, the next activity or next activities are executed, but only if the conditions are correct.

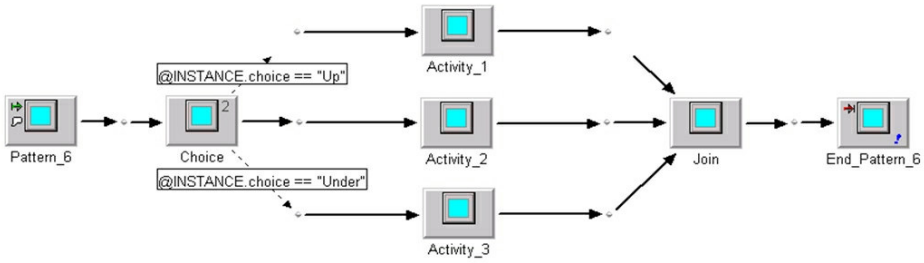



Fig. 6 Multi choice

***Support***

The multi choice pattern is supported by COSA.

The multi choice pattern can be implemented directly, whilst in addition, one can also specify conditions on the transitions themselves in COSA.

***Importance of the pattern***

Several choices or determinations are possible in executing separate flows. From the start of each process, it is possible to make more complex decisions, for instance when there are several parallel flows in the process. Multi choice prevents flows becoming too complicated and enhances a clear organization, with an orderly division of roles.

## Pattern 7: Synchronizing merge

***Description***

A point in the workflow process 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. This pattern assumes that that a branch that has already been activated, cannot be activated again while the merge is still waiting for other branches to complete. Synonym: synchronizing join.

***Example***

- Extending the example of Pattern 6 (Multi-choice), after either or both of the activities *Activity_1* and *Activity_3* have been completed (depending on whether they were executed at all), the activity *Join* needs to be performed (exactly once).
- Extending the example of the multi choice pattern (fig 6), after either or both activities have been completed (depending on whether the were executed at all), the path ends. So, a path ends irrespective of the number of paths that have been taken after the split.

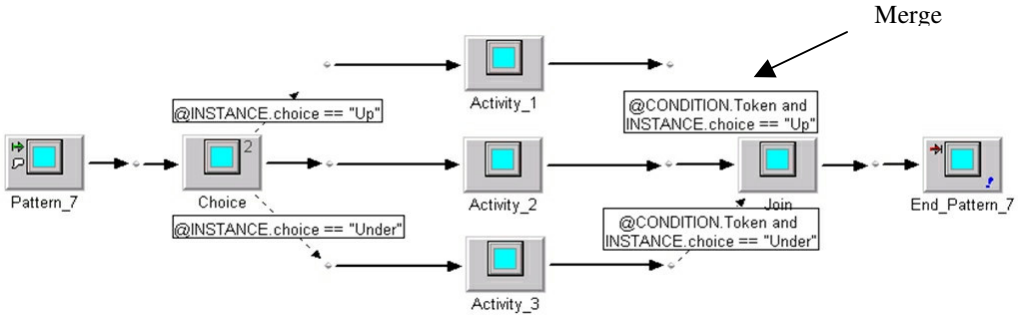



Fig. 7 Synchronizing Merge

***Support***

The synchronizing merge pattern is supported by COSA.
COSA uses its own methodology, i.e. using conditions. By using certain conditions, some paths can be excluded. Conditions at the end of the multi choice are used to complete the merge.

COSA Workflow 4.2 (Q1/2003) will also support 'condition labelling'. Condition rules can be given a logical name, making the model clear and so reducing the maintenance load.

***Importance***

It is important to use this pattern for controlling the merging of different paths.

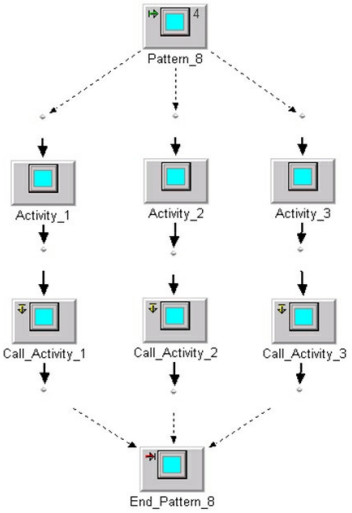
## Pattern 8: Multi merge description

***Description***

A point in a workflow process where two or more branches reconverge without synchronization. If parallel flows have parts in common, multi merge is needed. If more than one branch is activated either singularly or concurrently, the activity after the merge is restarted for every incoming branch.

***Example***

Sometimes two or more parallel branches share the same ending. Instead of replicating this (potentially very complicated) process for every branch, a multi-merge can be used. A simple example of this would be three activities *Activity_1, Activity_2 and Activity_3* running in parallel which should all be followed by an activity *close_case.*




The 'Call' activities all invoke the same subprocess with an activity *Close_case* that leads to the end of the pattern

Fig. 8 Multi Merge

***Support***

The multi merge pattern is supported by COSA.
COSA supports this pattern with reusable subflows; when paths share an ending, the same subflow is invoked.
COSA WfM offers the possibility to invoke sub-processes, executed synchronically. The main process waits until every sub-process is ready.
Since this can be considered a workaround, as per COSA Workflow 4.2 (Q1/2003) we will support multiple tokens in 1 place thus creating a direct support as described.

***Importance of the pattern***

Through the multi merge it is possible to control the model by simplifying the process. Important to this pattern is The synchronization of the sub-processes with the main process is essential for this pattern. Moreover it is important that variables can be exchanged between the processes.

## Pattern 9: Discriminator

***Description***

The discriminator is a point in a workflow process that waits for one of the incoming branches to be completed before activating the subsequent activity. From that moment on, it waits for all the remaining branches to be completed and "ignores" them. Once all incoming branches have been triggered, it resets itself so that it can be triggered again (which is important, otherwise it could not really be used in the context of a loop).

***Example***

To improve query response time, a complex search is sent to two different databases over the internet. The first one that comes up with the result should continue the flow. The second result is then ignored.

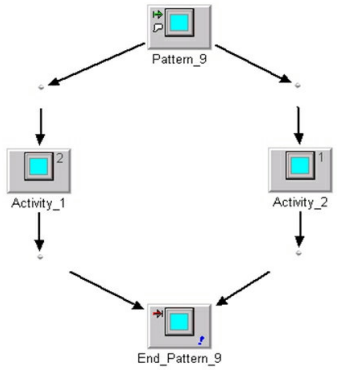



Fig. 9 Discriminator

***Support***

The discriminator pattern is supported by COSA.

To prevent the case in which a second instance of an activity is not generated due to the fact that the first instance is still active, COSA uses the 'skip' functionality. The 'skip' functionality allows activities within different paths to be ignored and be 'skipped' automatically if needed. The synchronisation (AND-join) ensures that following activities are only executed once. The exclusive choice (pattern number 4) is also a possibility, but the command *COSA_skip* can support the process in more complex situations.

***Importance of the pattern***

The discriminator makes it possible to make a choice between different activities. Whilst executing one activity, another one can be cancelled.

## Pattern 10: Arbitrary cycles

***Description***

A point in a workflow process where one or more activities can be carried out repeatedly. Synonyms: loop, iteration, cycle.

***Example***

The end of a path can be reached, when activity 1 is executed and handled correctly. Through the use of more variables and conditions a loop can have more or less complex appearances.

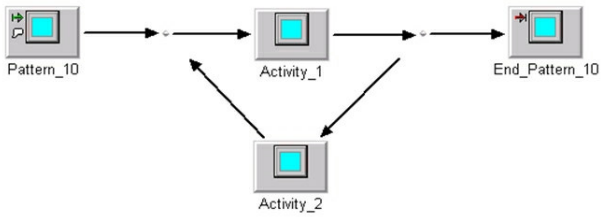



Fig. 10 Arbitrary Cycles

***Support***

COSA supports the arbitrary cycles pattern.

***Importance of the pattern***

If one activity is not executed correctly, the activity or parts of the activity must be repeated.

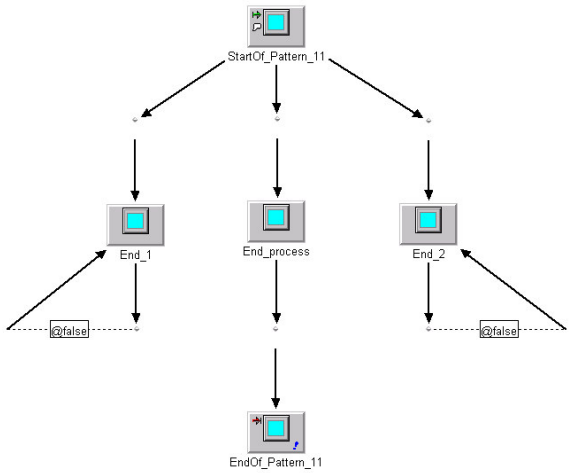
## Pattern 11: Implicit termination

***Description***

A given sub-process should be terminated when there is nothing else to be done. In other words, there are no active activities in the workflow and no other activity can be made active (and at the same time the workflow is not in deadlock).

***Example***

This semantics is typically assumed for every workflow model at the analysis stage




This model is made under the assumption that only one process model is allowed to be used (no sub processes can be called) The process has one end activity to terminate the process but multiple ending activities. When such ending activity is executed, the 'token' is consumed (i.e. it is disappeared from the flow). The end activity that terminates the process is only enabled when all ending activities are executed.

Fig. 11 Implicit Termination

***Support***

COSA supports the implicit termination pattern.

The use of ending activities that consume the token enables multiple ending points in one process model. When subprocesses are allowed to be used, this pattern can best be made with a-synchronous sub processes. These subprocesses are started from the main process, but run independent of the main process and have their own end activity.

The use of ending activities will only be needed when it is impossible to add an explicit ending to the process. This will only happen in very complicated processes, where validation for terminating a case is of utmost important.

With COSA Workflow 4.2 (Q1/2003) we will support multiple end modes in a process, thus having a direct support for this pattern.

***Importance of the pattern***

To avoid that activities, that happen to be running at the same time as a process is being terminated i.e. when an explicit Final node is reached, COSA supports the implicit termination pattern. The implicit termination pattern works in the same way as the explicit termination, with respect to controlling the end of the process.

## Pattern 12: Multiple instance without synchronization

***Description***

Within the context of a single case (i.e., workflow instance) multiple instances of an activity can be created, i.e., there is a facility to spawn off new threads of control. Each of these threads of control is independent on other threads. Moreover, there is no need to synchronize these threads. Synonyms: Multi threading without synchronization, spawn off facility.

***Example***

A customer ordering a book from an electronic bookstore may order several books at the same time. Many of the activities (e.g., billing, updating customer records etc.) occur at the level of the order. However, within the order multiple instances need to be created to handle the activities related to order each individual book (e.g., update stock levels, shipment, etc.) If the activities at the book level do not need to be synchronized, this pattern can be used.

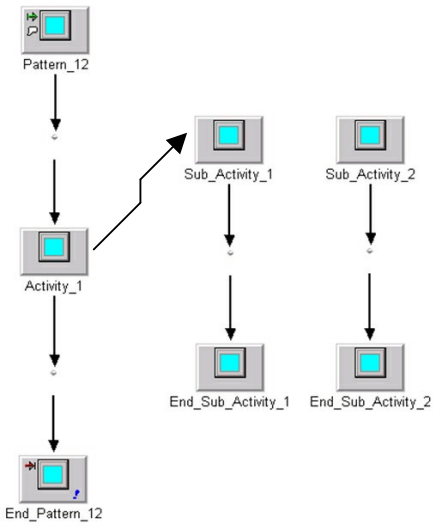



Fig. 12 Multiple Instances without

***Support***

COSA supports the multiple instances without synchronization pattern.

COSA supports an extra construct that enables the designer to create a sub-process or a subflow that will "spawn-off" from the main process and will be executed concurrently. One workflow may contain multiple concurrent flows that are created and share information. COSA offers a function to start multiple independent flows, contained together in a folder. By using triggers it is possible to restart more than one process within one folder. This folder connects processes, so information can be shared. You do not have to take the number of processes into account when you are modeling within a folder

***Importance of the pattern***

A very important aspect of workflow is to start asynchronous processes and coordinating them properly. The problem is not to *generate* multiple instances, the problem is to *coordinate* them and share information.

The next two patterns are based on this pattern; the simplest case is when we know, during the design of the process, the number of instances that will be active during process execution. In fact, this situation can be considered to be a combination of patterns 2 (parallel split) and 3 (synchronization) where all concurrent activities share a common definition.

## Pattern 13: Multiple instance with priori design knowledge

***Description***

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

***Example***

The requisition of hazardous material requires three different authorizations. If the number of instances is known a priori during design time, a very simple implementation option is to replicate the activity in the workflow model preceding it with a construct used for the implementation of the parallel split pattern. Once all activities are completed, it is simple to synchronize them using a standard synchronizing construct (Fig. 13).

Pattern 14 and 15 consider the situation where it is possible to determine the number of instances to be started *before* any of these instances is started.

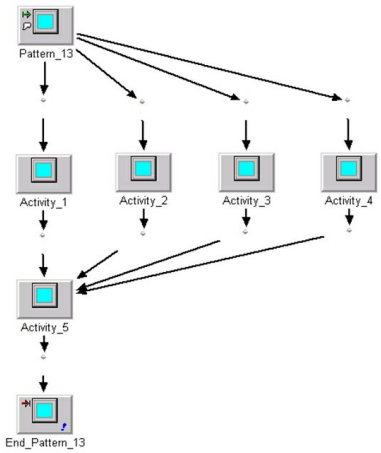



Fig. 13 Multiple Instances with priori  Design time knowledge

***Support***

COSA supports the multiple instance with priori design knowledge pattern.

***Importance of the pattern***

Only direct support for multiple threads within a model creates straightforward support. Any workaround will create a maintenance problem.

## Pattern 14: Multiple instance with priori runtime knowledge

***Description***

For a particular case, an activity is performed multiple times. The number of instances of a given activity for the case varies and may depend on characteristics of the case or availability of resources, but is known at some stage during runtime, before the instances of that activity have to be created. Once all instances are completed, another activity needs to be started.

***Examples***

- In the review process of a scientific paper submitted to a journal, the activity *review_paper* is initiated (instantiated) several times depending on the content of the paper, the availability of referees, and the credentials of the authors. The process is continued only if after all the reviews have been returned.
- When booking travel arrangements, the activity *book_flight* is executed multiple times if the trip involves more than one flight. Only after all the bookings have been made, is the invoice sent to the customer. Fig 14 is an example of this pattern; synchronizing happens through a 'rendez-vous' in the sub-process.

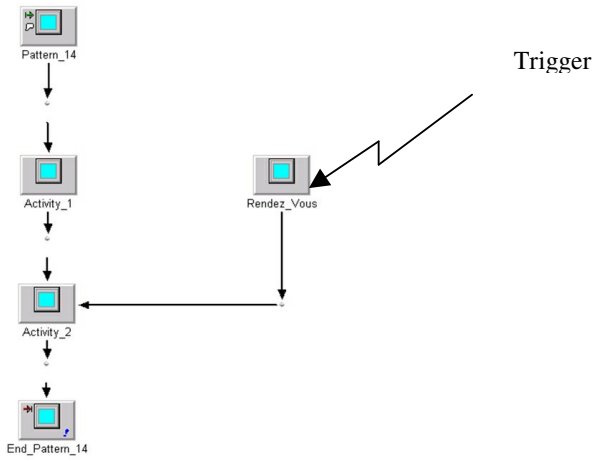



Fig. 14 Multiple Instances with priori runtime knowledge

***Support***

COSA supports the multiple instances with the a priori runtime knowledge pattern.
By using triggers, multiple sub-processes can be started if required. The number of sub-processes can be determined through variables (runtime knowledge). Like with pattern 12, information is shared through a common folder.

***Importance of the pattern***

During the process it is known how many processes (activities) have to be started asynchronically. The process is modelled in such a way that an asynchrone process can be started by means of a trigger, at any time in the process.

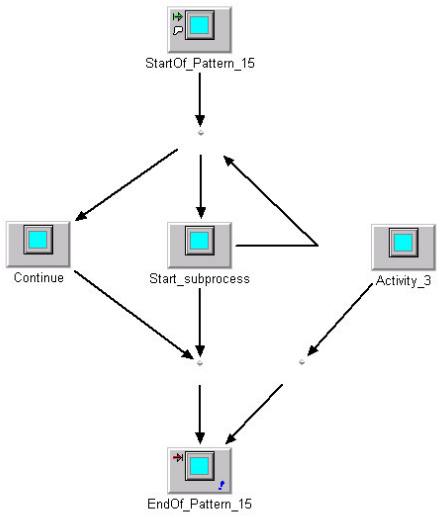
## Pattern 15: Multiple instance without runtime knowledge

***Description***

For a specific case, an activity is enabled multiple times. The number of instances of a given activity for the case is not known during design time, nor is it known at any stage during runtime i.e. before the instances of that activity have to be created. Once all instances are completed, some other activity 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.

***Examples***

- The requisition of 100 computers involves an unknown number of deliveries. Due the fact that the number of computers per delivery is unknown, the total number of deliveries is therefore also not known in advance. After each delivery, it can be determined whether a next delivery is to come by comparing the total number of delivered goods so far with the number of the goods requested. After processing all deliveries, the requisition has to be closed.
- For the processing of an insurance claim, zero or more eyewitness reports should be handled. The number of eyewitness reports may vary. Even when processing eyewitness reports for a given insurance claim, new eyewitnesses may surface and the number of instances may change.




Within a case (represented by a COSA folder), the Start_subprocess activity in the main process initiates subprocesses by the use of a trigger. This activity can be repeated as many times as needed, enabling an unlimited number of subprocesses. The number of started subprocesses in the folder is known as a case variable, thereby enabling the correct synchronisation of all active instances of the subprocesses before ending the main process.

Fig. 15 Multiple Instances without priori runtime knowledge

***Support***

COSA supports the multiple instance without a priori runtime knowledge.

***Importance of the pattern***

This pattern takes care of a well maintainable solution for asynchrone sub-processes.

## Pattern 16: Deferred choice

***Description***

A point in the workflow process where one of several branches is chosen. In contrast to the XOR-split, the choice is not made explicitly (e.g. based on data or a decision) but several alternatives are offered to the environment. However, in contrast to the AND-split, only one of the alternatives is executed. This means that once the environment activates one of the branches, the other alternative branches are withdrawn. It is important to note that the choice is delayed until the processing in one of the alternative branches is actually started, i.e. the moment of choice is as late as possible. Synonyms: External choice, implicit choice, deferred XOR-split.

***Examples***

- At certain points during the processing of insurance claims, quality assurance audits are undertaken at random by a unit external to those processing the claim. The occurrence of an audit depends on the availability of resources to undertake the audit, and not on any knowledge related to the insurance claim. Deferred Choices can be used at points where an audit might be undertaken. The choice is then between the audit and the next activity in the processing chain. The audit activity triggers the next activity to preserve the processing chain.
- After receiving products there are two ways to transport them to the department. The selection is based on the availability of the corresponding resources. Therefore, the choice is deferred until a resource is available.

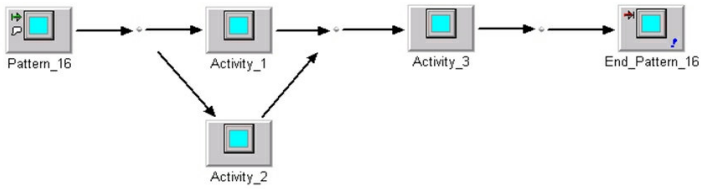



Fig. 16 Deferred Choice

***Support***

COSA supports the deferred choice pattern.

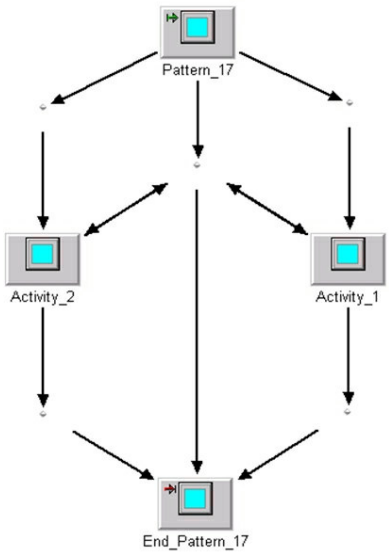
***Importance of the pattern***

This pattern is best when when a choice has to be taken implicitly (automatically by the system, based on an *event*), rather than explicitly (as decided manually by a user). In many business processes, this situation is the case.

## Pattern 17: Interleaved parallel routing

*Description*
A set of activities is executed in an arbitrary order: each activity in the set is executed, the order is decided during run-time, and no two activities are executed at the same moment. Synonym: unordered sequence.

*Examples*
- The Navy requires every job applicant to take two tests: *physical_test* and *mental_test.* These tests can be conducted in any order but not at the same time.
- At the end of each year, a bank executes two activities for each account: *add_interest* and *charge_credit_card_costs.* These activities can be executed in any order. However, since they both update the account, they cannot be executed at the same time.




In this figure 17 we can see, that if *activity_1* is being executed, *activity_2* doesn't have enough 'tokens' to start a different activity. At the moment that *activity_1* has been executed, a token occurs in the extra situation, by which *activity_2* can be executed.

COSA Workflow offers an additional solution to pattern 17: the COSA Activity Manager. The activity manager can make choices during modeling, between the activities that have to be started. The sequence of execution is not relevant.
It is also possible to mark activities as being obliged to execute or not.

Fig. 17 Interleaved Parallel Routing

***Support***
COSA supports the interleaved parallel routing pattern.
COSA supports this model both as a setting and as a model.

***Importance of the pattern***
Pattern 17 is important when several activities have to be started, but cannot be executed at the same time.

## Pattern 18: Milestone

***Description***

The execution of a certain activity depends on the status of the main process. An activity is only being executed if a certain milestone has been reached, which at the same time has not expire yet. Consider three activities named *activity_1*, *activity_2* and *activity_3*. *Activity_1* is only enabled if activity *activity_2* has been executed and *activity_3* has not been executed yet, i.e. *activity_1* is not enabled before the execution of *activity_2* and *activity_1* is not enabled after the execution of *activity_3*. Figure 18 illustrates the pattern. The state in between *activity_2* and *activity_3* is modelled by place *m*. This place is a milestone for *activity_1*. Note that *activity_1* does not remove the token from *m*: it only tests the presence of a token. Synonyms: Test arc, deadline, state condition, withdraw message.

***Examples***

- In a travel agency, flights, rental cars and hotels may be booked as long as the invoice is not printed.
- A customer can withdraw purchase orders until two days before the planned delivery.

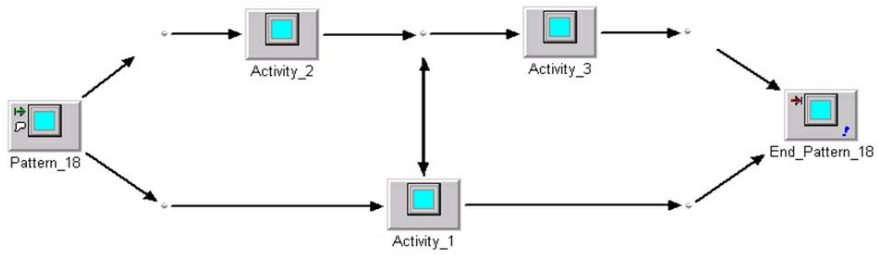



Fig. 18 Milestone

***Support***

COSA supports the milestone pattern.

The Petri Net technique that COSA Workflow uses to model processes makes this pattern very easy to support.

***Importance of the pattern***

The execution of some activities may disable others. A milestone can be used to test whether some part of the process is in a given state.

## Pattern 19: Cancel activity

***Description***

An enabled activity is disabled, i.e. a thread waiting for the execution of an activity is removed. Synonym: withdraw activity.

***Examples***

- Normally, a design is checked by two groups of engineers. However, to meet deadlines it is possible that one of these checks is withdrawn to be able to meet a deadline.
- If a customer cancels a request for information, the corresponding activity is disabled.

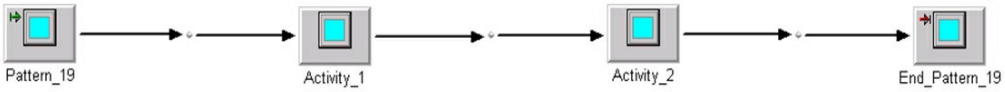



Fig. 19 Cancel Activity

***Support***

COSA supports the cancel activity pattern.

COSA WfM can create several patterns as cancel activities, by moving certain tokens or by generating virtual tokens. This function is a command that can be executed at each point in the process by an authorised user.

***Importance of the pattern***

The cancellation of an activity can be needed during a process. Pattern 19 solves this problem.

## Pattern 20: Cancel case

***Description***

A case, i.e. workflow instance, is removed completely (i.e., even if parts of the process are initiated/instantiated multiple times, all descendants are removed).

***Examples***

- In the process for hiring new employees, an applicant withdraws his/her application.
- A customer withdraws an insurance claim before the final decision is made.

***Support***

COSA supports the cancel case pattern.
COSA supports this by offering a control-flow that can trigger the withdrawal of all activities in the workflow at any time. This activity can be secured through authorisation for certain users.

***Importance of the pattern***

If at any time during the process one decides to cancel the workflow case, this pattern offers the solution.

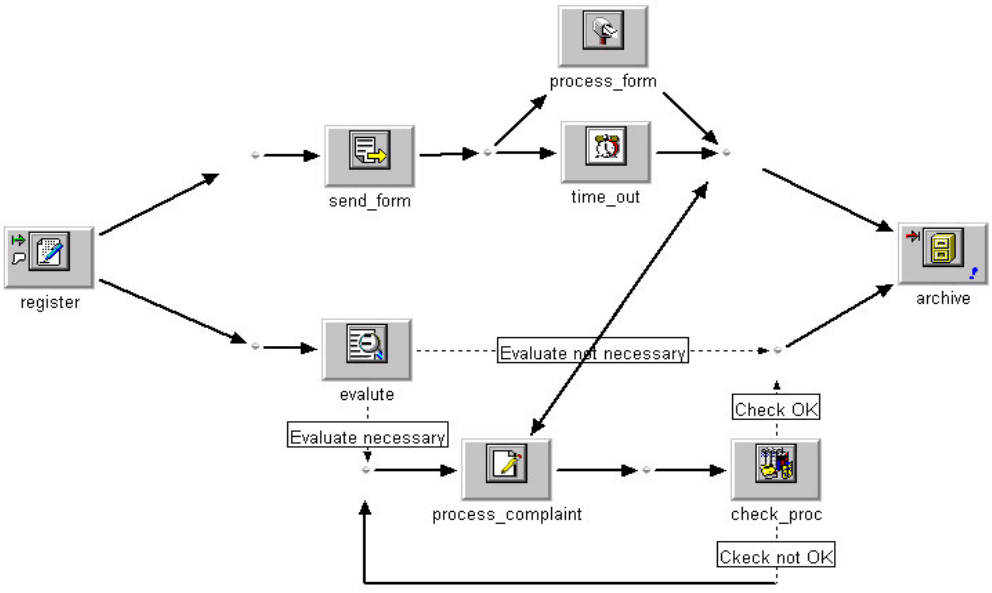
## Case Van der Aalst

Each year travel agency Y has to process a lot of complaints (about 10.000). There is a special department for the processing of complaints (department C). There is also an internal department called logistics (department L) which takes care of the registration of incoming complaints and the archiving of processed complaints. The following procedure is used to handle these complaints.

An employee of department L first registers every incoming complaint. After registration a form is sent to the customer with questions about the nature of the complaint. An employee of department C does this. There are two possibilities: the customer returns the form within two weeks or not. If the form is returned, it is processed automatically resulting in a report that can be used for the actual processing of the complaint. If the form is not returned on time, a time-out occurs resulting in an empty report. Note that this does not necessarily mean that the complaint is discarded.

After registration, i.e., in parallel with the form handling, the preparation for the actual processing is started. First, the complaint is evaluated by a complaint manager of department C. Evaluation shows that either further processing is needed or not. Note that this decision does not depend on the form handling. If no further processing is required and the form is handled, the complaint is archived. If further processing is required, an employee of the complaints department executes the task 'process complaint' (this is the actual processing where certain actions are proposed if needed). For the actual processing of the complaint, the report resulting from the form handling is used. Note that the report can be empty. A complaint manager checks the result of task ´process complaint´. If the result is not OK, task ´process complaint´ is executed again. This is repeated until the result is acceptable. If the result is accepted, an employee of the department C executes the proposed actions. After this the processed complaint is archived by an employee of department L.

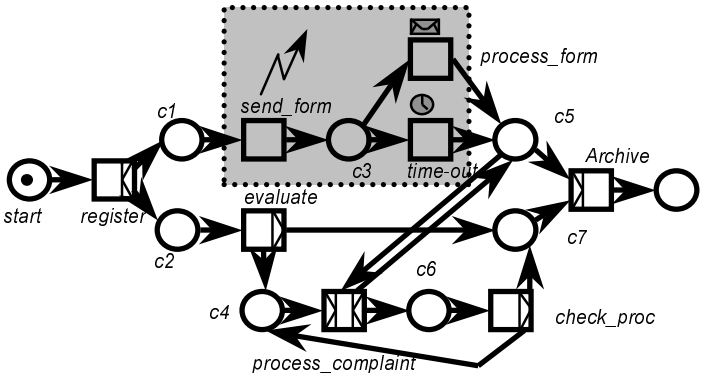
This example can be modelled in COSA Workflow as the following:



The model of this case contains two workflow patterns: the deferred choice and the milestone pattern. COSA is the only workflow supplier that supports these patterns completely, based on the mathematical Petri-net methodology.

This model shows the deferred choice pattern. The pattern makes sure that the activity after the deferred choice is being executed only if there hasn't occurred a time-out (through the time-out activities). The moment of choice (the deferred choice) in this pattern is essential. Considering the complaint handling case, a complaint can only be handled within the period that the form has been sent back.

**Deferred Choice**




In the complaint handling case, the milestone pattern assures that when a given situation is reached ('a milestone') the activity *process_complaint* can be triggered.

Within a parallel process, interaction is possible through the milestone pattern.

**Milestone**