Workflow patterns modelled in Arena

Jansen - Vullers, M.H.; Ijpelaar, R.E.A.; Loosschilder, M.W.N.C.

Published: 01/01/2006

Document Version
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Workflow patterns modelled in Arena

M.H. Jansen-Vullers
R. IJpelaar
M. Loosschilder

Table of contents

I. Introduction

This document describes briefly how to translate certain workflow patterns into the simulation language Arena. The patterns used for translation are described in [1] and [2]. We refer to these references to learn more about workflow patterns. Because of the complexity of the simulation program Arena, we refer to [3] to learn more about Arena. This document is understandable without the use of books or websites, but a bit of experience with Arena would be preferable.

Every modelling language has its own specific advantages and disadvantages. Arena has one major disadvantage when modelling workflow patterns; the way it deals with variables and attributes. The next chapter will give an explanation with an example to make this point clear.

This document describes six groups of workflow patterns that have been translated. Each group consists of a certain number of patterns with an own specific behaviour. The six groups are:

1. Basic control patterns
2. Advanced branching and synchronization patterns
3. Structural Patterns
4. Patterns involving multiple instances
5. State-based patterns
6. Cancellation patterns

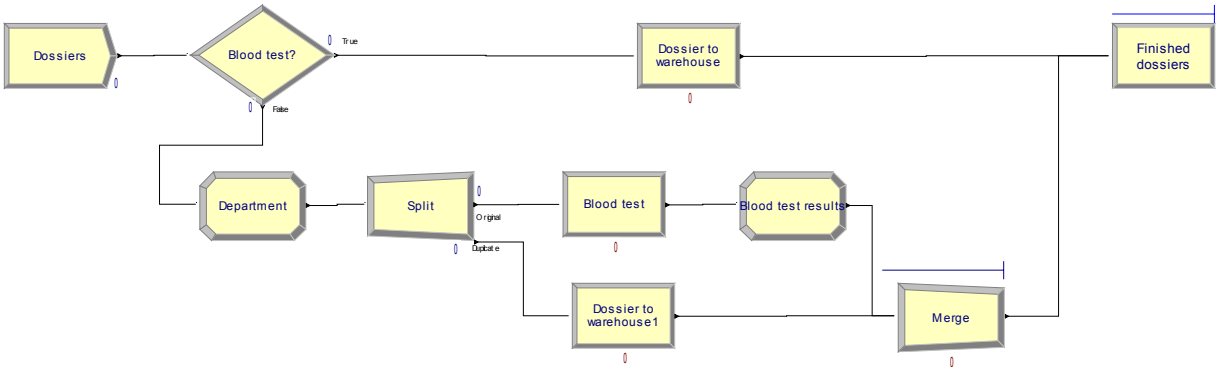
For each workflow pattern a short description will be given. We used the Petri net notation to give an example of the patterns and refer to [2] for a flash animation of each pattern that shows the behaviour of each pattern in a very clear and understandable way. Finally each pattern will end with an explanation how to model the pattern in Arena. A short description will be given how the pattern can be modelled or why the patterns cannot be modelled in a straightforward manner in Arena. Then the graphical notation in Arena will be shown and tables with the settings for each building block will make this document reliable in a sense that it is repeatable.

After the translation of the individual patterns, a real life case study will be modelled in Arena. The case study used, is described in [4, page 179]. The workflow and properties of the workflow including the resources will be given, and the direct translation of this workflow in Arena will be described. Finally a conclusion will be derived.

1. Variables and Attributes in Arena

In Arena, it is possible to assign attributes to entities and to use variables. The difference between variables and attributes is that when an attributes gets assigned to an entity it becomes a characteristic of every entity in the system. Variables are not entity specific and can be used or referred to at every moment in time.
Arena has its own special way of treating attributes and variables and it is very important to understand this, especially in cases where streams in workflows split or merge. To make sure that this explanation becomes not to abstract we use the following example:

In a hospital some clients need to do a blood test and the result of this test will be added to the client's dossier at a different department where the dossier will be stored. This means that part of the dossier will go to the blood test department and when the blood values are collected and reported, the dossier will be merged with the result of the blood values and the dossier can be stored. The dossiers that don't have to wait for a blood test will be stored right away. 50 percent of the clients need to undergo a blood test. The Arena model is shown in the following figure.



Dossiers go from block Dossiers to Blood test?, at Blood test? 50% goes directly to the warehouse where it gets stored and the other 50% will undergo a blood test (XOR split). At the Department block, an attribute with the department number will be assigned to a dossier. Then the Split block creates a dummy that is identical to the original entity (with the same attributes). After the process block Blood test the results of the blood test will be assigned and after that the block Merge merges the two streams into one. After that the dossiers will be stored at the block Finished dossiers.

Now the problem with the attributes in Arena can be stated. At the block Split two identical entities will be created and both get the attribute Department (let's say number 20). The original entity flows through the process block "Blood test" and gets a value assigned. All the entities in the system will get an attribute named Department with an initial value of 0. This is not really a problem when simulating but it is not very elegant. The real problem is the block "Merge". At this block two entities with the same serial number (automatically created by Arena) will be merged into one. At this block it is possible to do four things with the assigned attributes:

1. First
   The merged entity will get the values of the first entity entering the "Merge" block.
2. Last
   The merged entity will get the values of the last entity entering the "Merge" block.
3. Sum
   The merged entity will get the sum of the values of both entities.
4. Product
   The merged entity will get the product of the values of both entities.

None of these four possibilities is satisfying. This because of the attribute Department; the merged attributes can only get the right values if one of the first two possibilities (First or Last) will be used, and that it is sure and known that the entity that followed the path Blood test and Blood test results will constantly be the first or the last.

Because of the fact that defined variables are valid for the whole system, it is very important to know which variable will be used where, before using it in certain building blocks. Pattern 14 shows that it is very important to understand how Arena deals with variables and attributes.

For small cases with small process steps it is not a real problem but for larger and complex cases it can go wrong very easily. Thus consideration is required when using attributes or variables in complex modelling issues.

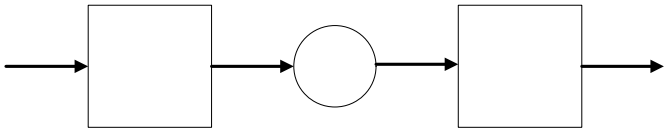
It is important to know that there are already a number of standard attributes in Arena. The problem described above can be avoided by only using these attributes. Arena does handle these attributes in the correct way when merging different flows. It is not always possible to model cases with the use of only the standard defined attributes, that's why this chapter has been added.

2. Basic Control Patterns

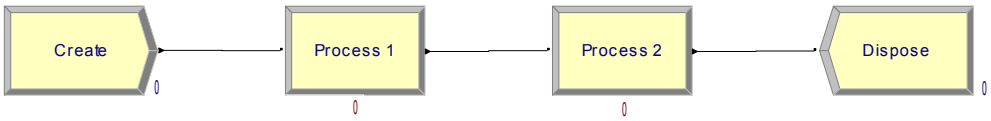
2.1 Pattern 1: Sequence

Description pattern
An activity in a workflow process is enabled after the completion of another activity in the same process.

Example



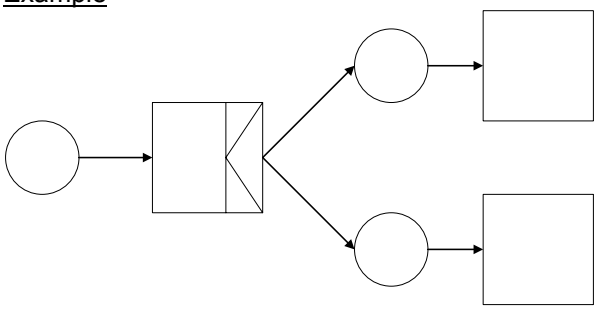
Translation to Arena
An entity will be created at the building block Create and undergo two activities (first Process 1 and then Process 2) and after the two process the entity will be destroyed. The building blocks Process 1 and Process 2 can be part of a complex system.



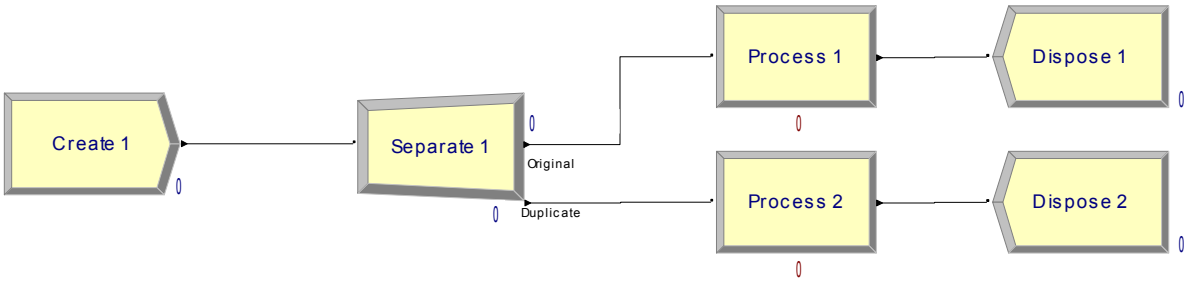
2.2 Pattern 2: Parallel Split

Description pattern
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 executed simultaneously or in any order.

Example



Translation to Arena
An entity will be created at Create 1. This entity will be duplicated at the block Separate 1 and the two entities (with the same serial number) undergo two different or equal processes. The block Separate 1 needs special attention for this pattern. It is necessary to fill in the form attached to the block Separate 1 and it is essential to use the right settings for this pattern.



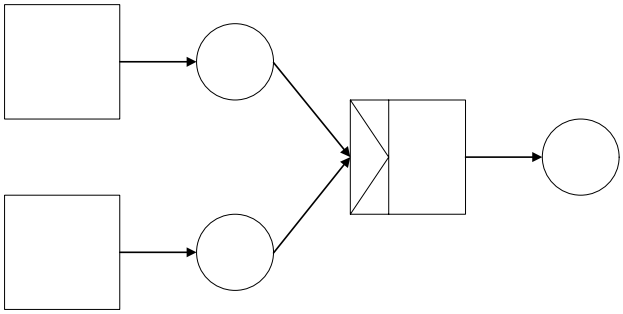
The settings for the block **Separate 1**:

Field	**Setting**
Name	User defined (dependent on the process)
Type	Duplicate Original
Percent cost to duplicate (0-100)	Cost of duplication (not included in this analyse)
# of Duplicates	1

2.3 Pattern 3: Synchronization

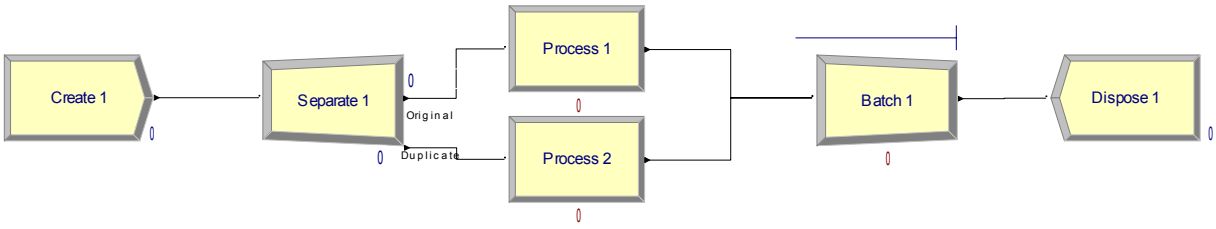
Description pattern
A point in the workflow process where multiple parallel subprocesses/activities converge into one single thread of control, thus synchronizing multiple threads. It is an assumption of this pattern that each incoming branch of a synchronizer is executed only once.

Example



Translation to Arena

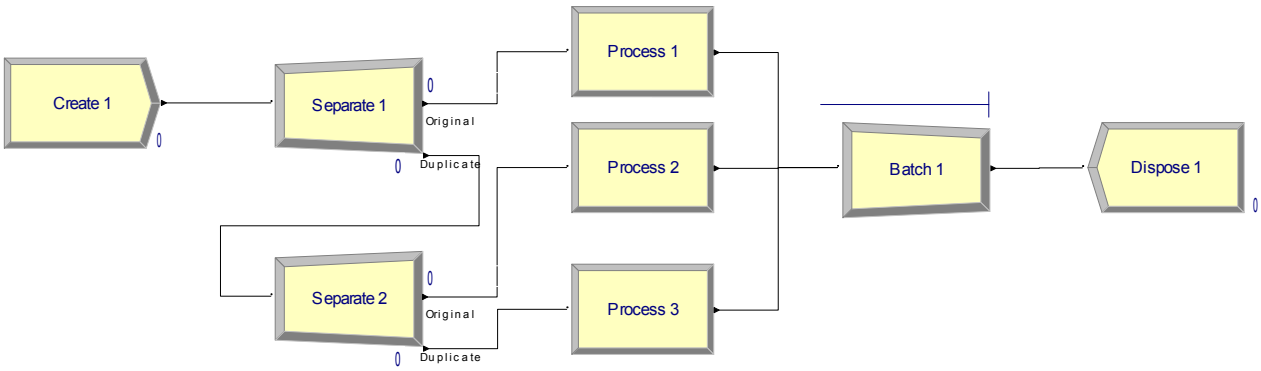
In most cases one process will be split into two paths. After a process or several processes the paths come together and are merged. Each created entity gets a serial number assigned by Arena. With the block Separate 1 it is possible to create two entities with the same serial number. At the block batch it is possible to merge entities with the requirement that they have the same serial number. A disadvantage of the block Batch 1 is that it is not possible to use more than one requirement; only one attribute can be specified.



The settings for the block **Batch 1**:

Field	**Setting**
Name	User defined (dependent on the process)
Type	Permanent
Batch Size	2 (Number of entities combined)
Save Criterion	Sum (combines the attributes of the entities)
Rule	By Attribute
Attribute	Entity.SerialNumber

Merging more than two incoming streams can be done by placing more blocks of the type Separate in a sequence. The following figure illustrates this.

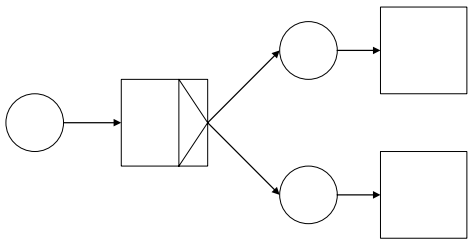


The setting Batch Size for the Batch block is 3 in this case. This value is the only value that shall change in this process and increases in proportion with an increase of the number of incoming streams.

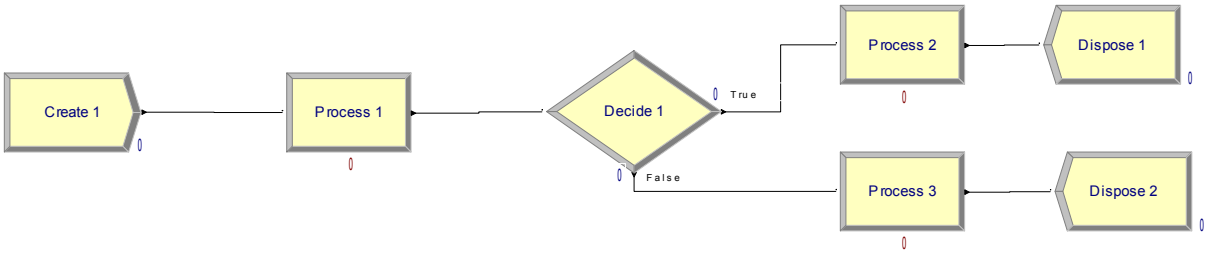
2.4 Pattern 4: Exclusive Choice

Description
A point in the workflow process where, based on a decision or workflow control data, one of several branches is chosen.

Example



Translation to Arena
The most important block for this pattern is the Decide 1. At this block it is possible to choose the conditions that decide which path an entity will follow (Process 2 or Process 3).



The settings for the block **Decide 1:**

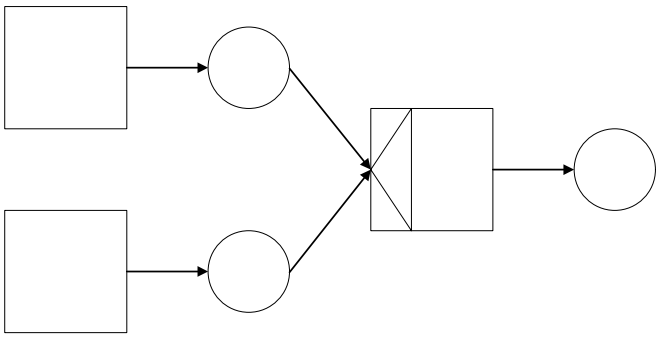
Field	**Setting**
Name	User defined (dependent on the process)
Type	All options are possible dependent on the process

With more than two possible paths to continue after Decide 1, it is possible to use the setting “N-way by chance” or “N-way by condition” in the field Type.

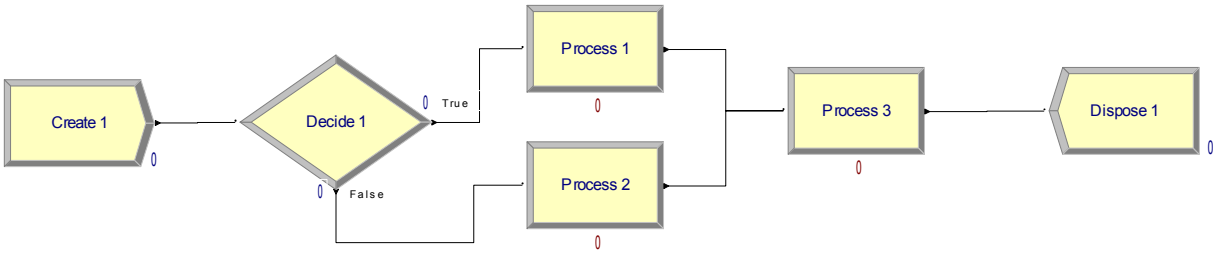
2.5 Pattern 5: Simple Merge

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 (if this is not the case, then see Pattern 8 (Multi-merge) or Pattern 9 (Discriminator)).

Example



Translation to Arena
Because of the assumption that Process 1 and Process 2 can never be executed in parallel, the block Decide 1 sends an entity with a certain chance or condition to either Process 1 or Process 2. This makes sure that only two paths can be followed:
Process 1 – Process 3 or
Process 2 – Process 3

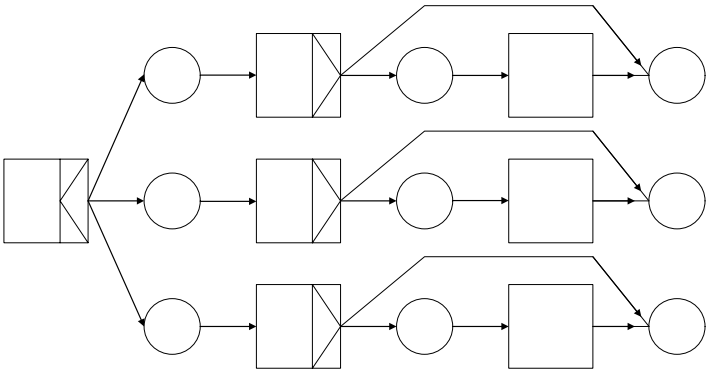


3. Advanced Branching and Synchronization Patterns

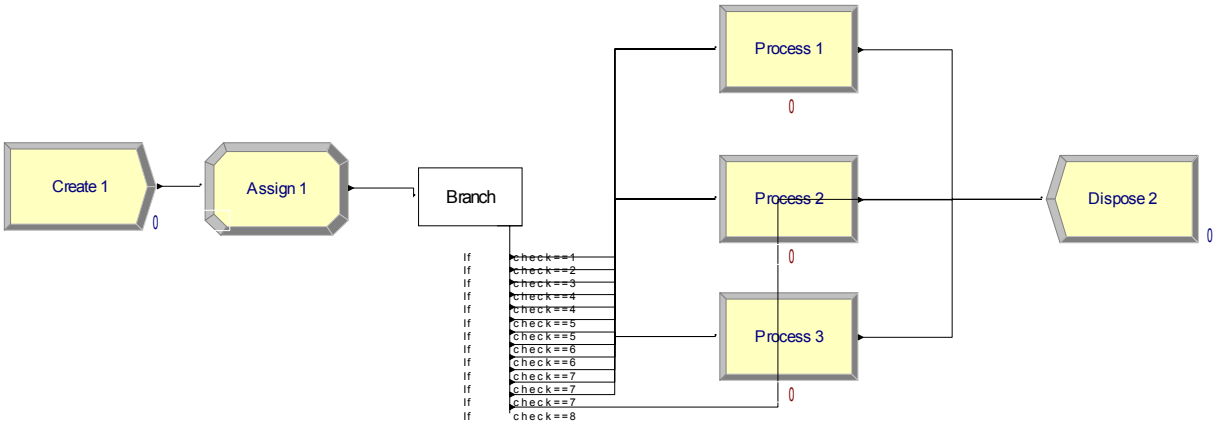
3.1 Pattern 6: Multi Choice

Description
A point in the workflow process where, based on a decision or workflow control data, a number of branches are chosen.

Example



Translation to Arena
Pattern 4, the exclusive choice, assumes that exactly one of the alternatives is selected and executed (**or** process 1 **or** process 2). The multi choice can be used to choose multiple alternatives from a given set of alternatives. For example if you want to book a hotel you can choose to rent a car, rent a bike or rent a boat or any possible combination of the three alternatives. In Arena it is possible to model this construct and it is shown in the figure below.
With three processes, there are eight possible ways to flow through the system (Process 1, Process 2, Process 3, Process 1 and Process 2, Process 1 and Process 3, Process 2 and Process 3, Process 1 and Process 2 and Process 3 and finally none of the three processes). After the creation of an entity, the entity gets an attribute assigned with a value between 1 and 8 (discrete probability function). The block “Branch” will send the entities to one of the eight possibilities depended on the value of the assigned attribute. The Branch block will duplicate entities if they have to go through two processes (rent a bike and a car but no boat).



The settings for the block **Assign 1**:

Field	**Setting**
Name	User defined (dependent on the process)
Type	Attribute
Attribute Name	check
New Value	DISC(1/8, 1,2/8, 2, 3/8 ,3, 4/8, 4, 5/8, 5 , 6/8, 6, 7/8 , 7,1,8) (Discrete probability function)

The settings for the block **Branch**:

Field	**Setting**
Label	No text
Max number of Branches	15
Branch Types	If, check == 1, ,Yes If, check ==2, , Yes : : if, check == 7, , Yes if, check == 7, , Yes if, check == 7, , Yes if, check == 8, , Yes

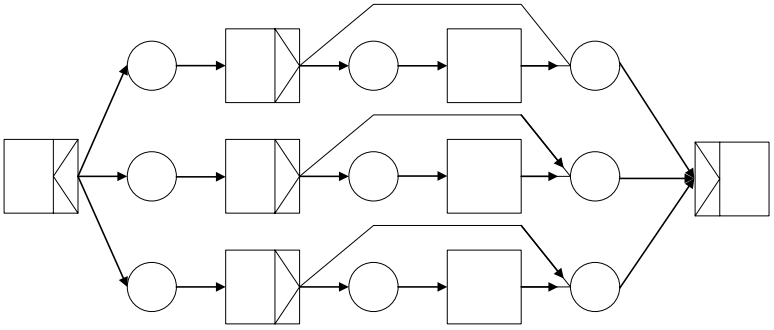
When an entity enters the block Branch the settings of the branch type will be sequentially checked. If check = 1 then the entity will go to Process 1. If check = 7 the entity will go to Process 1, Process 2 and Process 3, so the branch block will create two duplicates.

3.2 Pattern 7: Synchronizing Merge

Description

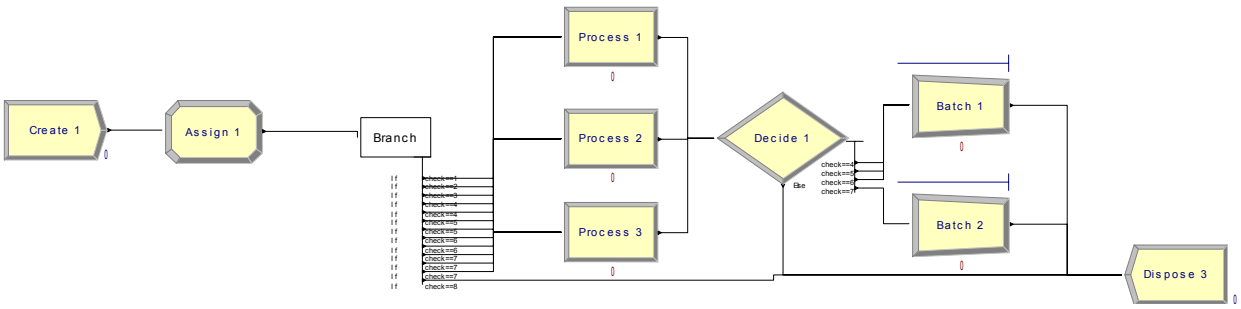
A point in the workflow process where multiple paths converge into one single thread. If more than one path is taken, synchronization of the active threads needs to take place. If only one path is taken, the alternative branches should reconverge without synchronization. It is an assumption of this pattern that a branch that has already been activated, cannot be activated again while the merge is still waiting for other branches to complete.

Example



Translation to Arena

In this pattern the entities that come from the processes 1, 2 or/and 3 will be merged. The first part of the figure below describes the multi choice explained in the previous pattern. For this pattern the blocks Decide 1, Batch 1 and Batch 2 have been added. Consider the previous pattern, if the attribute value check is 1,2 or 3 only one entity flows through the system and it is not necessary to batch. If the value is 4,5 or 6 there are two entities that will be batched in the block Batch 1. When the value is 7, three entities will be batched in the block Batch 2.



The blocks Branch, Process 1,2 and 3 have the same settings as in the previous pattern.

The settings for the block **Decide 1:**

Field	**Setting**
Name	User defined (dependent on the process)
Type	N-way by condition
Conditions	Attribute, check == 4 Attribute, check == 5 Attribute, check == 6 Attribute, check == 7

Decide has five outgoing channels as shown in the figure above. If the value of check is 1,2 or 3, the criteria entered in the settings above are not true and the entity will go directly from Decide 1 to the block Dispose 1.

The settings for the block **Batch 1**:

Field	**Setting**
Name	User defined (dependent on the process)
Type	Permanent
Batch Size	2 (Number of entities combined)
Save Criterion	Sum (combines the attributes of the entities)
Rule	By Attribute
Attribute	Entity.SerialNumber

The settings for the block **Batch 2**:

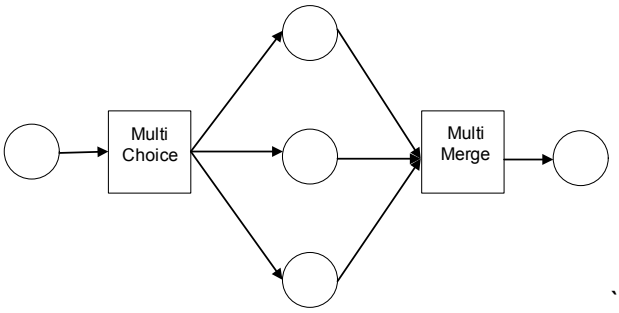
Field	**Setting**
Name	User defined (dependent on the process)
Type	Permanent
Batch Size	3 (Number of entities combined)
Save Criterion	Sum (combines the attributes of the entities)
Rule	By Attribute
Attribute	Entity.SerialNumber

Notice that the number of entities that have to be merged is the only difference between Batch 1 and Batch 2.

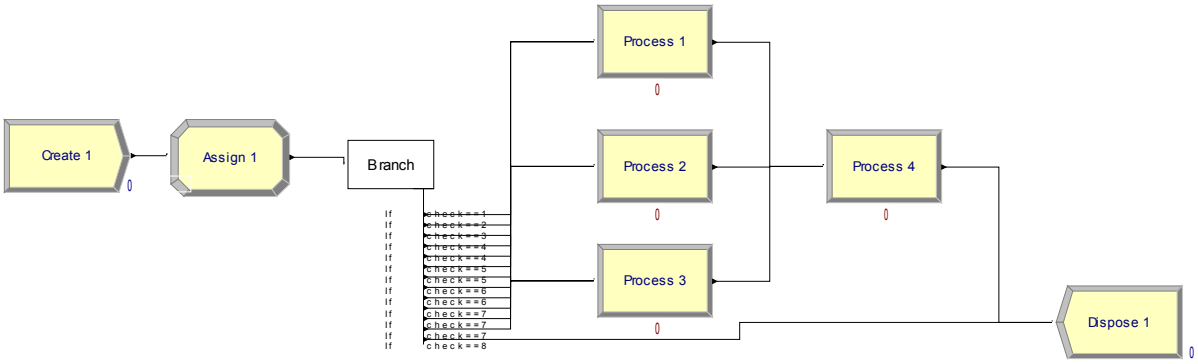
3.3 Pattern 8: Multiple Merge

Description
A point in a workflow process where two or more branches reconverge without synchronization. If more than one branch gets activated, possibly concurrently, the activity following the merge is started *for every activation of every incoming branch*.

Example



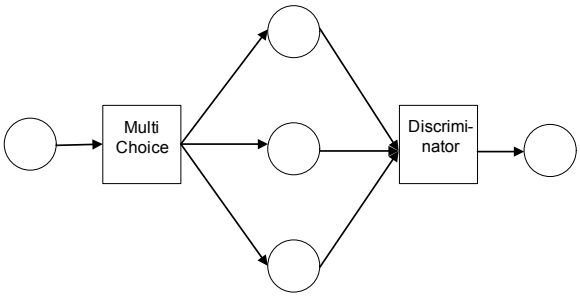
Translation to Arena
This pattern is the same as pattern 7 except for the synchronization step. Instead, after Process 1,2 or 3 an entity goes directly to Process 4 without synchronization. The figure below show the graphical notation in Arena.



3.4 Pattern 9: Discriminator

Description
The discriminator is a point in a workflow process that waits for one of the incoming branches to complete before activating the subsequent activity. From that moment on it waits for all remaining branches to complete 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

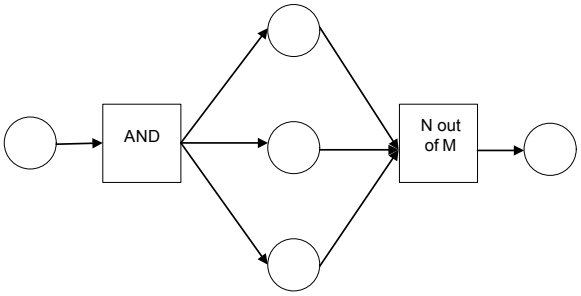


Translation to Arena
This construct has the property to model a N-out-of-M join, a multiple merge and a synchronizing merge. It is not possible to model the N-out-of-M join construct in a straightforward way. In Arena it is only possible to remove and “destroy” entities that are waiting in a predefined queue. Pattern 9a explains why this construct is not possible.
It is possible to merge two or more streams as described in pattern 8 without synchronization and pattern 7 with synchronization.

3.5 Pattern 9a: M out of N join

Description
N-out-of-M Join is a point in a workflow process where *M* parallel paths converge into one. The subsequent activity should be activated once *N* paths have completed. Completion of all remaining paths should be ignored. Similarly to the discriminator, once all incoming branches have "fired", the join resets itself so that it can fire again.

Example

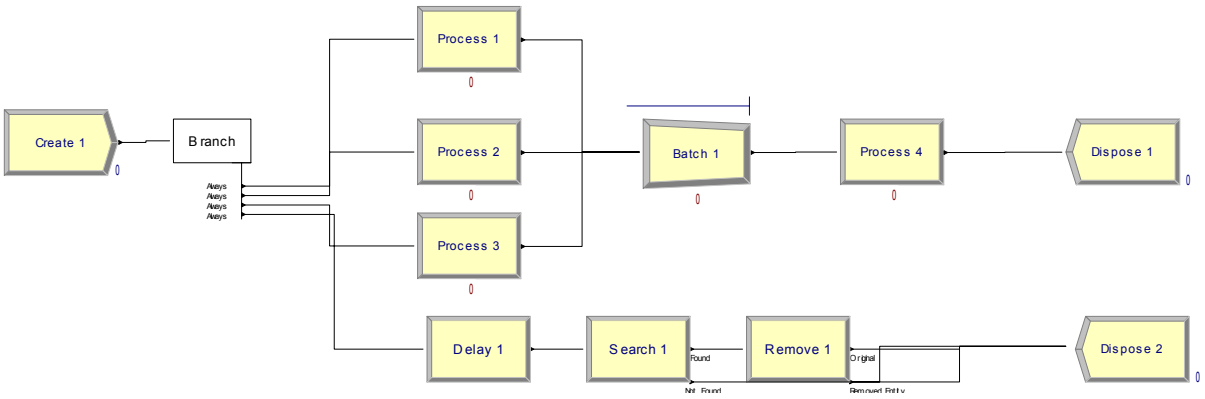


Translation to Arena

In workflows it is not “aloud’ to store entities in the system. If three entities arrive at the N-out-of-M block some of the entities have to be destroyed dependent on the settings of N and M. In Arena it is only possible to model N-out-of-M constructs in an elegant way using the following parameters:

N>1
M<2N-1

The figure below shows a 2 out of 3 system.



One entity enters the block Branch and the Branch block will make three duplicates of the entity and send them to Process 1,2 and 3 and to the block Delay 1. This is the first part of the construct (an AND split). In the second part the N-out-of-M part, the block Batch 1 batches N entities. Because the entities arriving at Batch 1 are duplicates, it is not possible to make a distinction between the entities. That is why a restriction for N is N>1. In the model design it is unknown which entity will arrive at first at the block Batch 1. It is not possible to send one entity to Process 4 and destroy the other entities. If N>1, lets say 2, it is possible to batch two entities and destroy the last entity that stays behind in Batch 1 (a vacuum cleaner effect). This is only possible if M>2N-1. For example an 2 out of 4 construct, the Batch 1 block will batch two times because there are four entities with the same serial number and the batch size is set to two. This is not desirable.
Thus, in a straightforward way it is only possible to model an N-out-of-M construct with N>1 and M<2N-1.

Now let’s go back to the 2 out of 3 system. To remove an entity from a queue, the blocks Delay 1, Search 1 and Remove 1 are necessary. The Delay block makes sure that an entity with a particular serial number has passed the blocks Process 1,2 and 3 and is stored in Batch 1(the two other entities have been batched into one entity and continue to Process 4). Then the Search 1 block searches the queue in Batch 1 for entities with the same serial number (is always the last entity that finished a process). The Remove 1

block makes sure that the entity will be removed from the queue in Batch 1. This has the effect of a vacuum cleaner, no entities will stay behind in the system.

The settings for the block **Batch 1**:

Field	**Setting**
Name	User defined (dependent on the process)
Type	Permanent
Batch Size	N=2 for the N-out-of-M join (Number of entities combined)
Save Criterion	Sum (combines the attributes of the entities)
Rule	By Attribute
Attribute	Entity.SerialNumber

The settings for the block **Delay 1**:

Field	**Setting**
Name	User defined (dependent on the process)
Allocation	Wait
Delay Time	Larger than the largest process time
Units	Minutes, Hours, Days.. etc.

The settings for the block **Search 1**:

Field	**Setting**
Name	User defined (dependent on the process)
Type	Search a queue
Queue Name	Batch 1. Queue
Starting Value	1
Ending Value	NQ (The whole queue)
Search condition	Entity.SerialNumber

The settings for the block **Remove 1**:

Field	**Setting**
Name	User defined (dependent on the process)
Queue name	Batch 1.Queue
Rank of entity	J

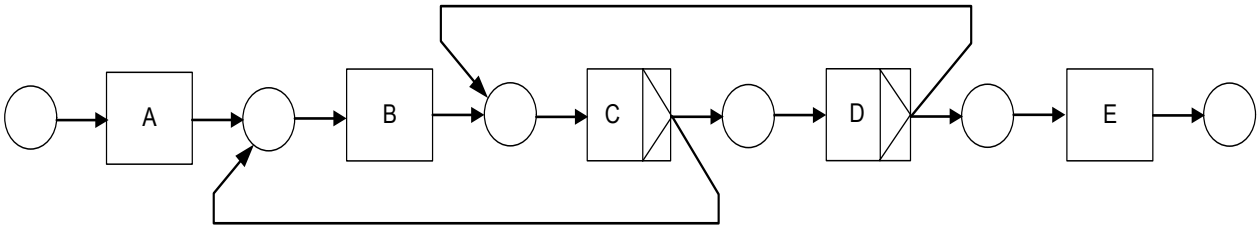
If the block search finds the entity where it is looking for the entity gets the rank J. The remove block will remove the entity ranked with J from the specified queue.

4. Structural patterns

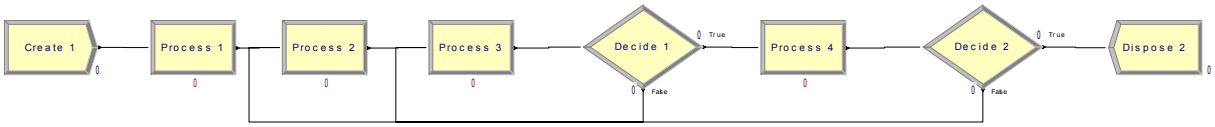
4.1 Pattern 10: Arbitrary Cycles

Description
A point in a workflow process where one or more activities can be done repeatedly.

Example



Translation to Arena
This pattern is very easy to model in Arena. After the execution of a process it is possible to decide which direction to go (to repeat a part of the process or not). The figure below shows Arena model.



An entity moves sequentially to Process 1,2 and 3 and after the processes the decision will be made to repeat Process 2 and 3 or to go on with Process 4. After Process 4 the decision will be made to repeat Process 3 or not. The decision can be based on a certain chance or on a specified condition.

4.2 Pattern 11: Implicit Termination

Description
A given subprocess 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).

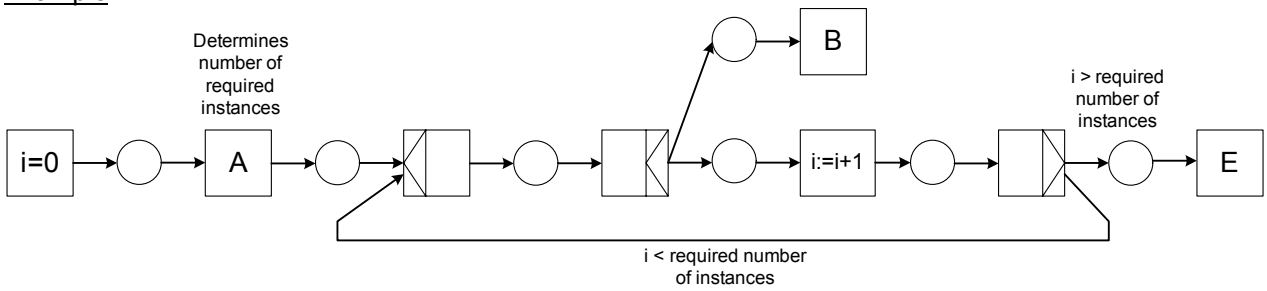
Translation to Arena
In Arena it is only possible to terminate the total simulation after a specified time. It is not possible to change settings when the simulation is running, neither is it possible to abort a process while running. Arena only makes use of explicit termination. For this reasons it is not possible to translate this pattern into Arena.

5. Patterns Involving Multiple Instances

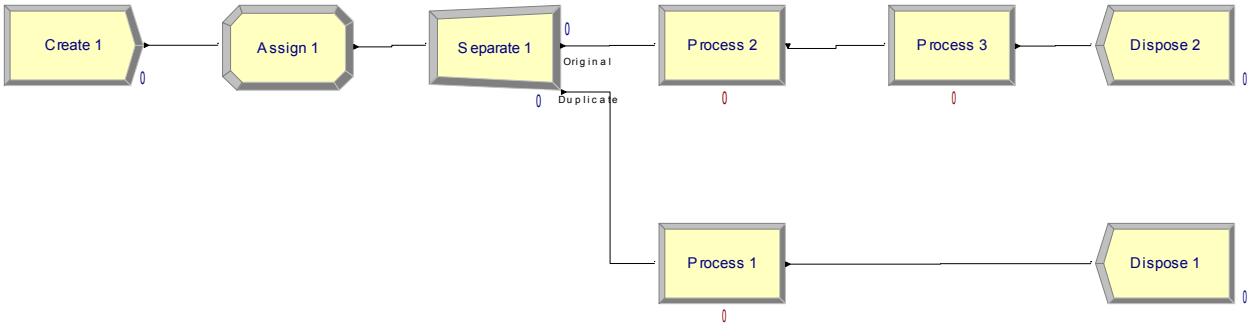
5.1 Pattern 12: Multiple Instances Without Synchronization

Description pattern
Within the context of a single case 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 of other threads. Moreover, there is no need to synchronize these threads.

Example



Translation to Arena
In this model multiple instances are created. The starting block Create 1 just creates one entity every specified creation period. In the block Assign 1 the number of instances that will be duplicated is determined and stored in the variable “NrInstances”. The Separate block creates multiple copies of the original entity. Process 1 is therefore executed NrInstances number of times. The original entity is sent through to Process 2 so it can proceed its way to Process 3 and the rest of the process. No synchronization is needed in this pattern.



The settings for the block **Assign 1**:

Field	**Setting**
Name	User defined (dependent on the process)
Type	Variable
Variable Name	NrInstances
New Value	Enter required number of instances or have ARENA generate the number of instances

The settings for the block **Separate 1**:

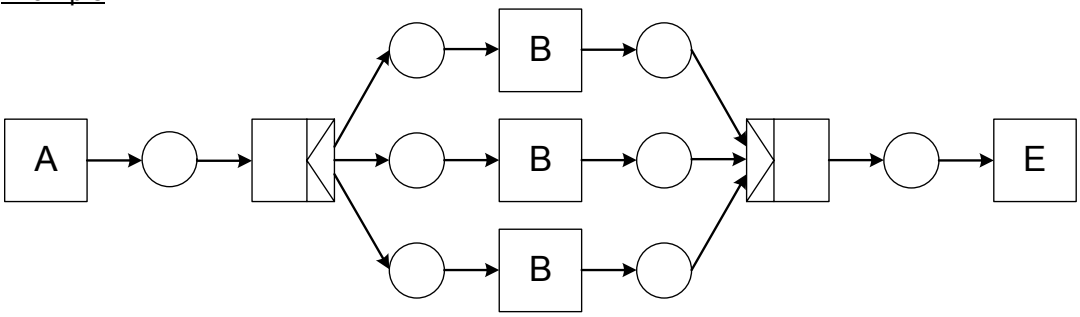
Field	**Setting**
Name	User defined (dependent on the process)
Type	Duplicate Original
Percent cost to duplicate (0-100)	Cost of duplication (not included in this analysis)
# of Duplicates	NrInstances

5.2 Pattern 13: Multiple Instances With a Priori Design Time Knowledge

Description pattern

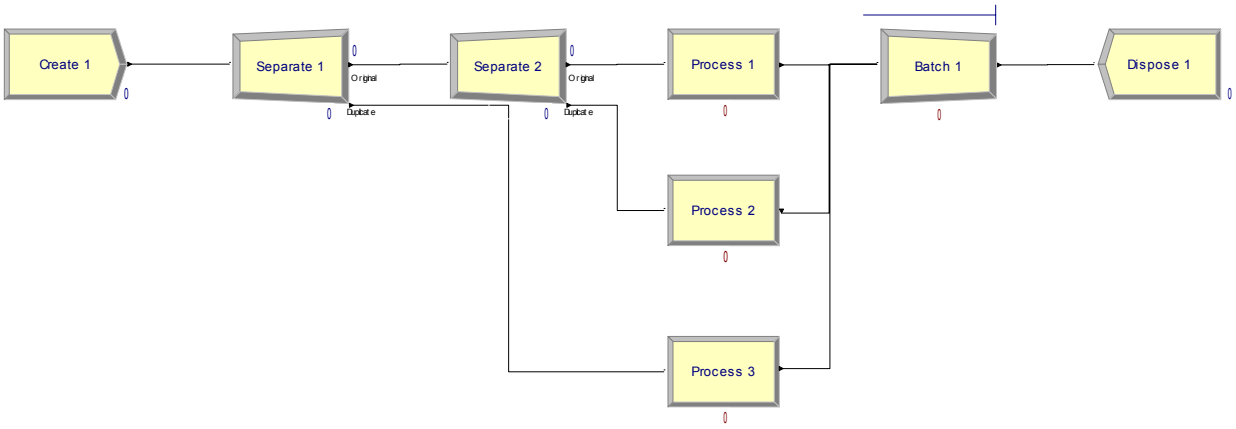
For one process instance an activity is enabled multiple times. The number of instances of a given activity for a given process instance is known at design time. Once all instances are completed some other activity needs to be started.

Example



Translation to Arena

For this model the number of instances is already known a priori during design time. Therefore the activities are replicated and put in different parallel branches. Once all activities are completed the multiple instances are synchronized in Batch 1. In this example it was known during design time that the number of instances was 3. Therefore three replicated braches are executed in parallel. To model a three way AND split it is necessary to use two sequential Separate blocks because these blocks can only realize a two way split or a Branch block. In block Batch 1 the instances are synchronized using batch by attribute. The entities will have to wait in Batch 1 until three entities arrive at Batch 1 with the same serial number. The separate blocks create copied entities with the same serial number as the original entity.



The settings for the blocks **Separate 1 and Separate 2**:

Field	**Setting**
Name	User defined (dependent on the process)
Type	Duplicate Original
Percent cost to duplicate (0-100)	Cost of duplication (not included in this analysis)
# of Duplicates	1

The settings for the block **Batch 1**:

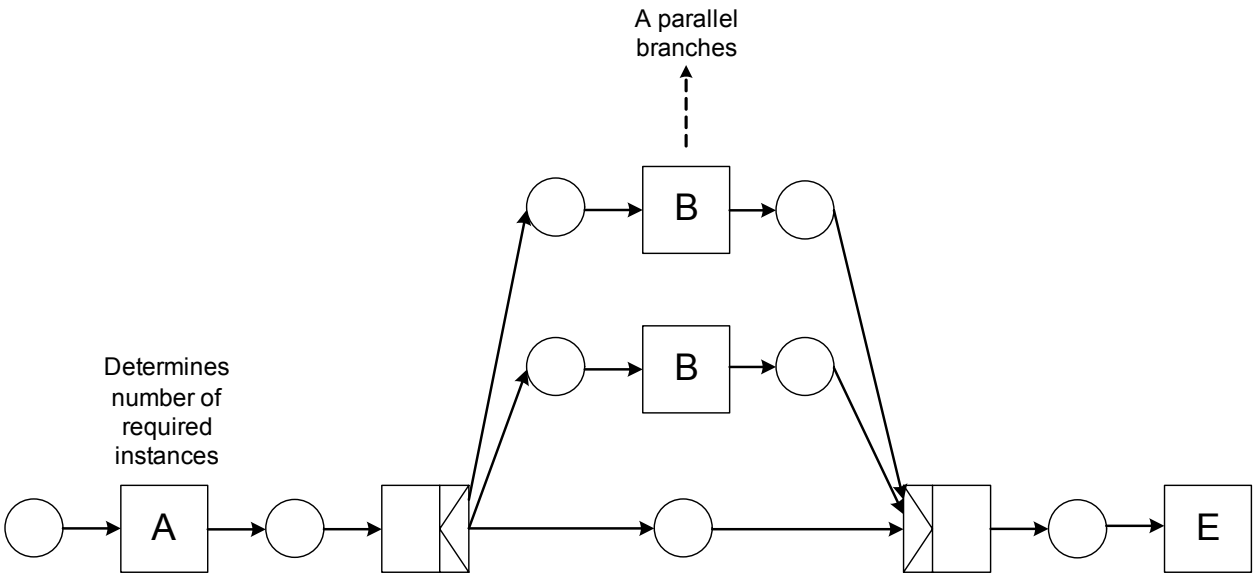
Field	**Setting**
Name	User defined (dependent on the process)
Type	Permanent
Batch Size	In this example 3 (Number of entities combined)
Save Criterion	Sum (combines the attributes of the entities)
Rule	By Attribute
Attribute Name	Entity.serialNumber

Note: the Arena equivalent of pattern 14 can also be used to model pattern 13. This way it is not necessary to replicate the branches.

5.3 Pattern 14: Multiple Instances With a Priori Run Time Knowledge

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

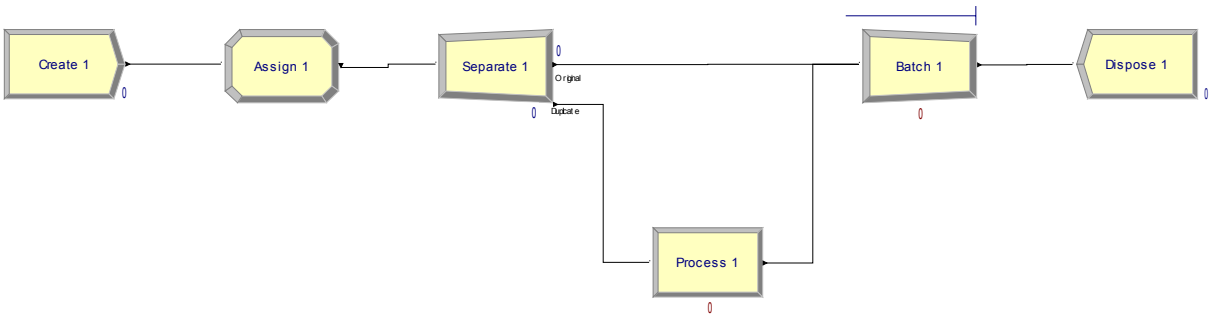
Example



Translation to Arena

In this pattern the number of instances is not known during design time but at some stage during runtime, before the instances are created. In this case is the number of instances stored in the variable "NrInstances", which is assigned to the entity in the block Assign 1. In Separate 1 the variable Nrinstances is checked and the original entity is duplicated the number of times that is specified in NrInstances. The original entity will continue its normal path and will wait in Batch 1. The duplicates will go through a parallel branch and will undergo Process 1. This processing of the duplicated entities will be in parallel, so none of the duplicates will have to wait for the other to get processed. When all duplicated entities are processed, the waiting original entity (in Batch 1) and all duplicates are batched permanently. The batched entity can then proceed its way through the process.

A big disadvantage of this model is that the original entity and the duplicates must be batched first before a new entity is allowed to execute Assign 1. When a new entity executes Assign 1 before the old entities are batched the new entity will change the value of the variable NrInstances, so Batch 1 will wait for the wrong number of entities when it is batching the old entities. Thus it has to be assumed that a new entity will never execute Assign 1 before all former entities are batched.



The settings for the block **Assign 1**:

Field	**Setting**
Name	User defined (dependent on the process)
Type	Variable
Variable Name	NrInstances
New Value	Number of Instances

The settings for the block **Separate 1:**

Field	**Setting**
Name	User defined (dependent on the process)
Type	Duplicate Original
Percent cost to duplicate (0-100)	Cost of duplication (not included in this analysis)
# of Duplicates	NrInstances

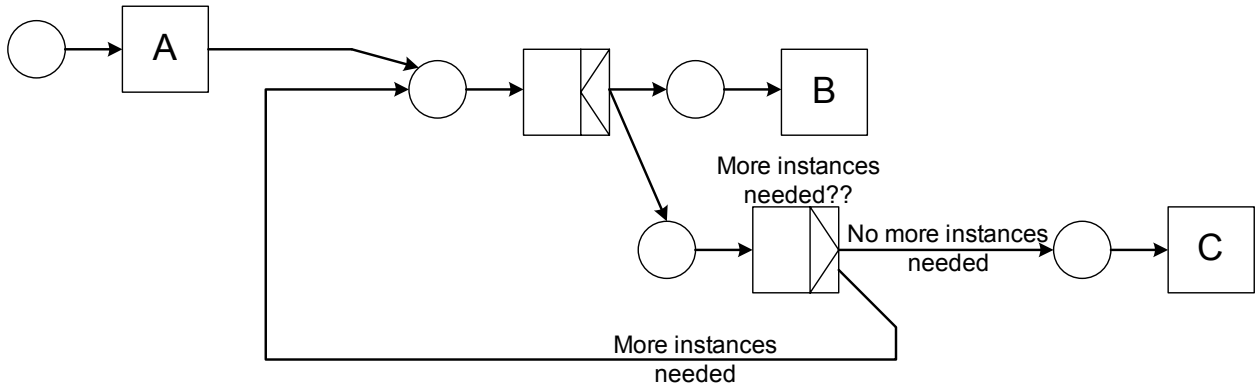
The settings for the block **Batch 1:**

Field	**Setting**
Name	User defined (dependent on the process)
Type	Permanent
Batch Size	NrInstances + 1
Save Criterion	Sum (combines the attributes of the entities)
Rule	By Attribute
Attribute Name	Entity.serialNumber

5.4 Pattern 15: Multiple Instances Without a Priori Run Time Knowledge

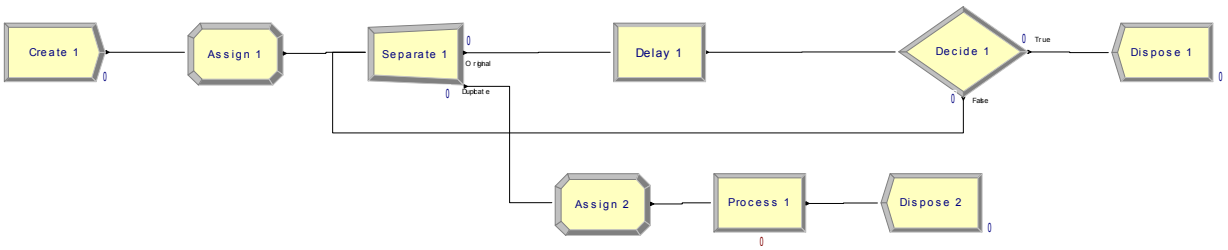
Description pattern
For one case an activity is enabled multiple times. The number of instances of a given activity for a given case is not known during design time, nor is it known at any stage during runtime, before the instances of that activity have to be created. Once all instances are completed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

Example



Translation to Arena
In this pattern is the number of required instances not known at design time but somewhere during runtime. The entities are created in the Create 1 block. The initial value for the variable NrInstances is set in the block Assign 1. This variable is used during runtime to verify whether more instances are needed. Assign 2 is the block that increases the number in the variable NrInstances. This value is checked in the Decide 1 block. When this value is lower then a certain value, the entity goes to Separate 1 and a new entity will undergo Process 1 and the number of instances increases again. The entity can continue to dispose when there are indeed enough instances in the system. The block Delay 1 makes sure that Separate 1 creates a duplicate. The separate block creates a duplicate after a small time step, because decide 1 doesn't consume time, no duplicate would be created.

This pattern has the same disadvantage as pattern 14. The same assumption has to be made here.



The settings for the block **Assign 1**:

Field	**Setting**
Name	User defined (dependent on the process)
Type	Variable
Variable Name	NrInstances
New Value	0

The settings for the block **Separate 1:**

Field	**Setting**
Name	User defined (dependent on the process)
Type	Duplicate Original
Percent cost to duplicate (0-100)	Cost of duplication (not included in this analysis)
# of Duplicates	1

The settings for the block **Assign 2**:

Field	**Setting**
Name	User defined (dependent on the process)
Type	Variable
Variable Name	NrInstances
New Value	NrInstances+x (with x the number of increased instances)

The settings for the block **Decide 1**:

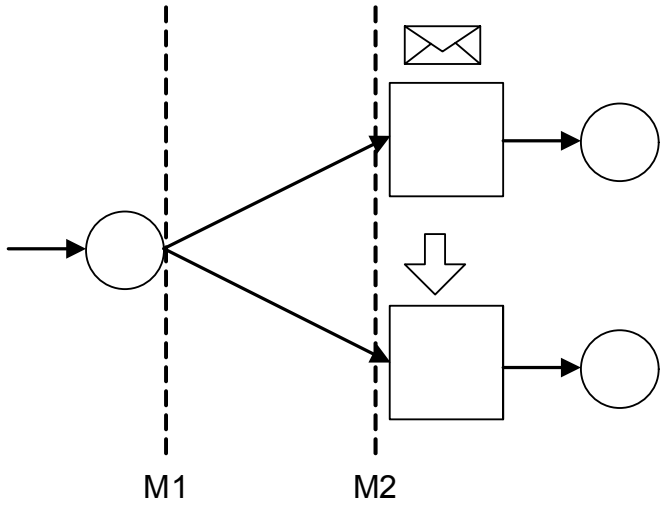
Field	**Setting**
Name	User defined (dependent on the process)
Type	2-way by condition
If	Variable
Named	NrInstances
Is	>=
Value	Minimal value of required NrInstances

6. State-based patterns

6.1 Pattern 16: Deferred Choice

Description pattern
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.

Example

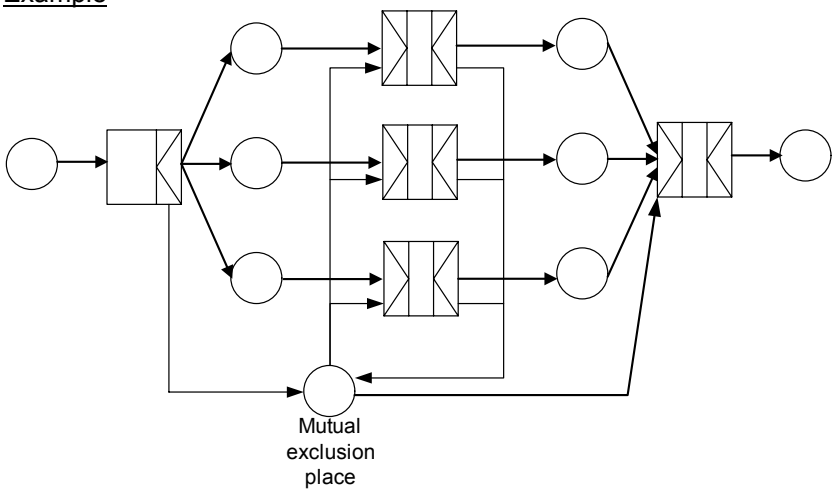


Translation to Arena
In this pattern is the moment of choice very important. The choice is not made in the place (at M1) but in the transitions (at M2). The entities/tokens are not pushed into the transitions. Instead a queue is formed in the place and the transitions pull an entity/token when one of the activities is triggered. The translation into Arena is a problem since the moment of choice in Arena is always in a decide block (as in pattern 4). There is no other way of choosing a path in Arena. Entities in Arena are always pushed into the next block. So Arena does not support this pattern.

6.2 Pattern 17: Interleaved Parallel Routing

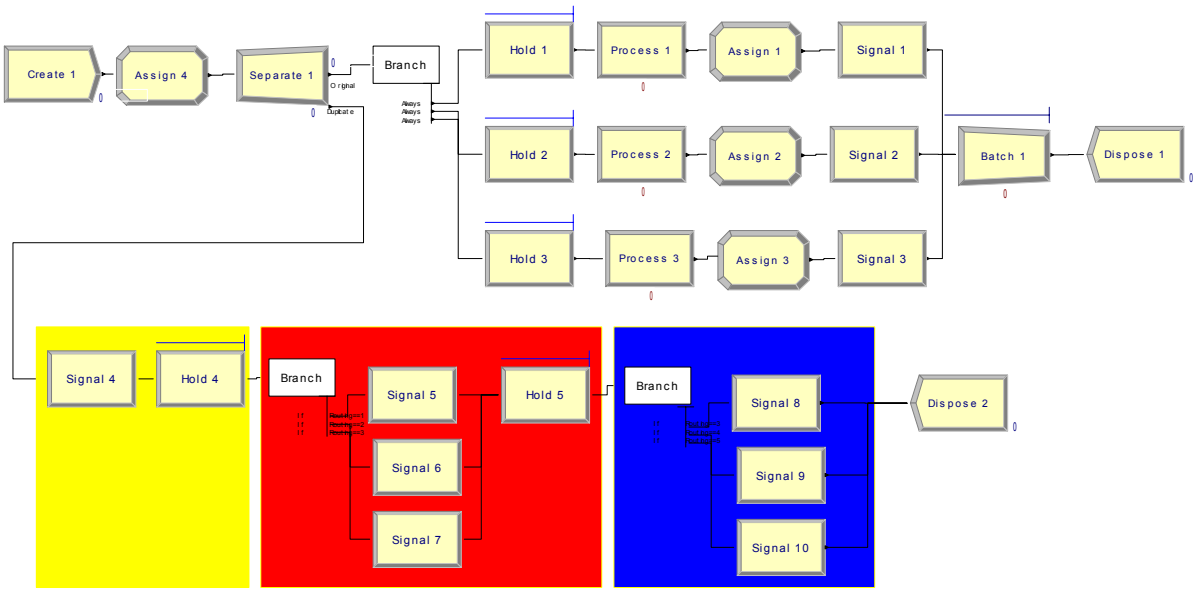
Description pattern
A set of activities is executed in an arbitrary order: Each activity in the set is executed, the order is decided at run-time, and no two activities are executed at the same moment (i.e. no two activities are active for the same workflow instance at the same time).

Example



Translation to Arena

This pattern is not directly supported by Arena. There is no block in Arena that models the desired behaviour. It is possible though to implement this pattern in Arena. The disadvantage is that the Arena model is complex. The blocks in the upper part of the model below depict the actual branches containing activities. The lower part of the model controls the entities in the upper part.
The first Branch block represents an AND split and sends entities to the three Hold blocks. Here they wait for a signal. An entity is duplicated and is sent to the control elements. The blocks in the yellow part decide what branch is chosen by randomly choosing 1, 2 or 3. It then signals the waiting entities and one branch is executed. The control entity continues its path when the first branch is executed. The blocks in the red part take care of the second branch that has to be chosen. It then sends a signal to the two remaining waiting entities and one of them executes a process in the chosen branch. Then finally the blocks in the blue part make sure that the remaining branch is executed. The three entities are batched after all three processes have been executed. The Assign blocks are used to alter the variables that are needed to route the entities. The same assumption as made in patterns 14 and 15 holds for this pattern.



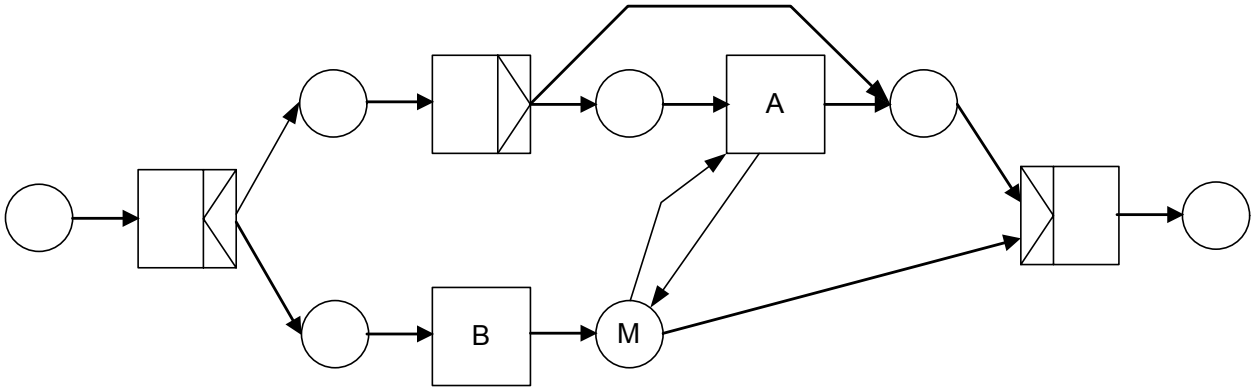
For the settings of the blocks used in this pattern is referred to the Arena file “Pattern 17.doe”.

6.3 Pattern 18: Milestone

Description pattern

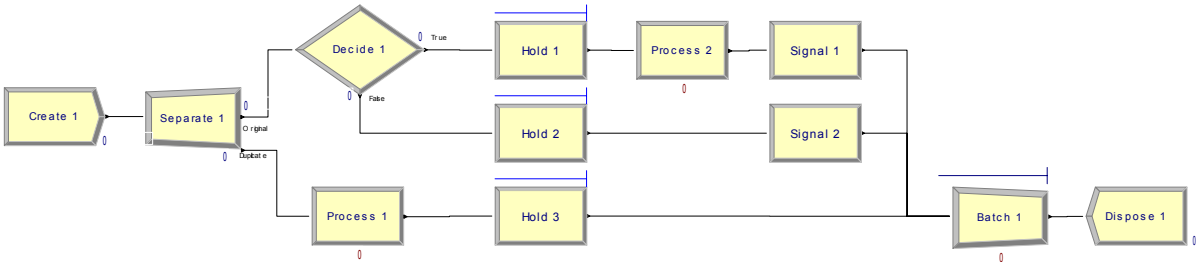
The enabling of an activity depends on the case being in a specified state, i.e. the activity is only enabled if a certain milestone has been reached which did not expire yet. Consider two activities named A and B. Activity A is only enabled if activity B has been executed, i.e. A is not enabled before the execution of B. The net in the example below illustrates the pattern. The state after B is modeled by place M. This place is a milestone for A. Note that A does not remove the token from M: It only tests the presence of a token.

Example



Translation to Arena

The milestone pattern can be translated into Arena. An entity that is created at Create 1 is duplicated and sent to two parallel branches. Process 1 is executed at all times. Before Process 2 there is a choice whether Process 2 has to be executed or not. The entity arrives at Hold 1 or Hold 2 after this choice has been made. It waits there until the entity in the lower branch has completed Process 1. This entity is then seized in Hold 3. The entity in Hold 1 or Hold 2 is released when there is an entity in Hold 3. Process 2 can be executed or skipped now. After this the entity in Hold 3 is released from its Hold block by a signal from one of the signal blocks. The two entities are then batched into one entity and the rest of the process can continue.



The settings for the block **Separate 1:**

Field	**Setting**
Name	User defined (dependent on the process)
Type	Duplicate Original
Percent cost to duplicate (0-100)	Cost of duplication (not included in this analysis)
# of Duplicates	1

The settings for the block **Decide 1**:

Field	**Setting**
Name	User defined (dependent on the process)
Type	dependent on conditions for skipping Process 2

The settings for the block **Hold 1 and Hold 2:**

Field	**Setting**
Name	User defined (dependent on the process)
Type	Scan for condition
Condition	NQ(Hold 3.Queue) == 1
Queue Type	Queue
Queue Name	Hold 1.Queue / Hold 2.Queue

The settings for the block **Hold 3:**

Field	**Setting**
Name	User defined (dependent on the process)
Type	Wait for signal
Wait for value	1
Queue Type	Queue
Queue Name	Hold 3.Queue

The settings for the block **Signal 1 and Signal 2:**

Field	**Setting**
Name	User defined (dependent on the process)
Signal value	1
Limit	1

The settings for the block **Batch 1:**

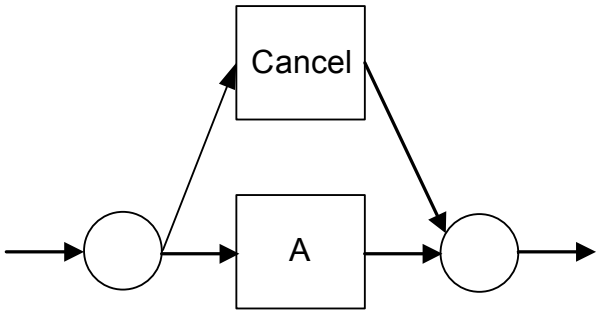
Field	**Setting**
Name	User defined (dependent on the process)
Type	Permanent
Batch Size	2
Save Criterion	Sum (combines the attributes of the entities)
Rule	By Attribute
Attribute Name	Entity.serialNumber

7. Cancellation Patterns

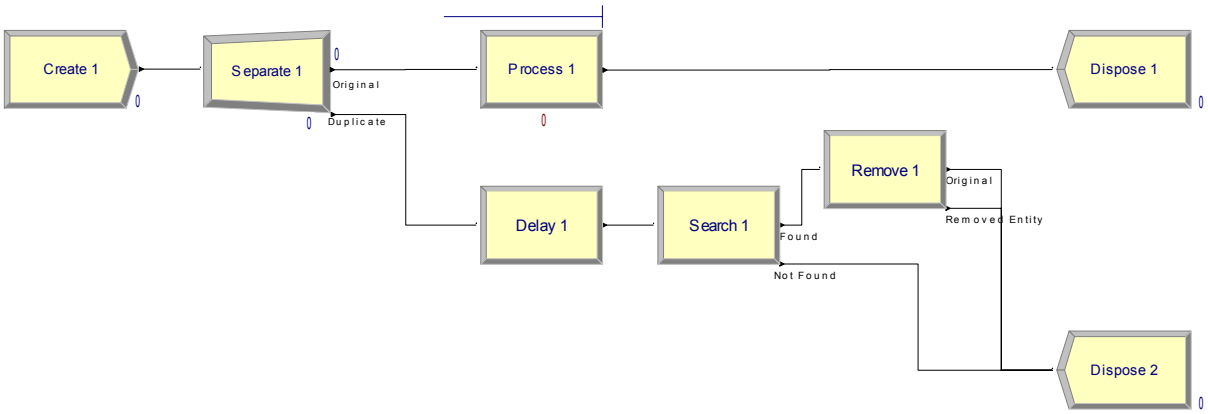
7.1 Pattern 19: Cancel Activity

Description pattern
An enabled activity is disabled, i.e. a thread waiting for the execution of an activity is removed.

Example



Translation to Arena
In Arena it is possible to remove an entity from a certain queue. The Arena model below shows an example where a case (an entity) is cancelled when it has to wait for more then 30 minutes in a queue. Another reason for cancellation can of course be used. An entity is created in Create 1 and is duplicated using a separate block. The original entity is sent to process 1 and the copied dummy entity is sent to the block Delay 1. In this block the dummy waits for 30 minutes. The dummy then continues its path and searches the queue in Process 1. When the dummy can find the entity with the same serial number it means that that entity is already waiting for 30 minutes. Remove 1 takes care of the actual removal of the original entity from the queue in Process 1. The search dummy is disposed when the entity with the specific serial number cannot be found.



The settings for the block **Separate 1:**

Field	**Setting**
Name	User defined (dependent on the process)
Type	Duplicate Original
Percent cost to duplicate (0-100)	Cost of duplication (not included in this analysis)
# of Duplicates	1

The settings for the block **Delay 1:**

Field	**Setting**
Name	User defined (dependent on the process)
Allocation	Other
Delay Time	30
Untis	Minutes

The settings for the block **Search 1:**

Field	**Setting**
Name	User defined (dependent on the process)
Type	Search a Queue
Queue Name	Process 1.Queue
Starting Value	1 (first entity in the queue)
Ending Value	NQ (last entity in the queue)
Search Condition	entity.serialnumber

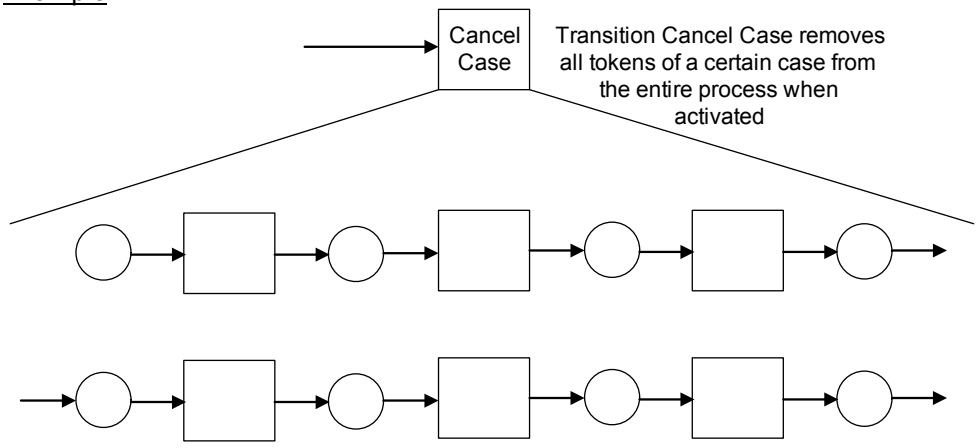
The settings for the block **Remove 1:**

Field	**Setting**
Name	User defined (dependent on the process)
Queue Name	Process 1.Queue
Rank of entity	J (result of the search block)

7.2 Pattern 20: Cancel Case

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

Example



Translation to Arena
In Arena it is not possible to remove all entities, representing a certain case, at once when this case is cancelled completely. It is possible in Arena to search in all queues at once for a certain entity but it is impossible to use just one remove block and remove the specified entities from all queues in the process. It is possible though to duplicate the construct used in pattern 19 for every single activity, but this solution is not very elegant.

8. De Grote Beek case

The “Grote Beek Case” as described in [4] is modelled and described below.

The workflow of the “Grote Beek Case” is first modelled in the Petri Net notation and can be found in appendix 1. The model also shows what workflow patterns have been used to model the workflow. It can be seen that workflow patterns 1-6 are used. All these patterns are directly supported by Arena. The Arena model was then build, based on the Petri Net model. The Arena model can be seen in appendix 2. The workflow patterns 1-6, described earlier can be distinguished from the model.

The process starts in block Create 1. New entities are created with a negative exponentially distributed inter arrival time of 25 minutes. All tasks in the process are modeled in Arena by using a Process block. The following tasks are performed and described in the model:

Task	Performed by	Service Time
Answer Notice	Secretarial Office Worker	Uniform (1,3)
Record Notice	Nurse Officer	Exponential (15)
Store and Print Notice	Secretarial Office Worker	Beta (5,15)
Create Patient File	Secretarial Office Worker	Uniform (3,7)
Close Case	Secretarial Office Worker	Beta (2,7)
Assign Intakers	Team Leader	Beta (2,8)
Store Assignment	Secretarial Office Worker	Beta (2,8)
Hand out Cards	Secretarial Office Worker	Uniform (1,3)
Ask for Medical File	Secretarial Office Worker	Beta (2,6)
Update Patient File	Secretarial Office Worker	Uniform (5,12)
Plan Meeting 1st Intaker	1st Intaker	Uniform (1,3)
Plan Meeting 2nd Intaker	2nd Intaker	Uniform (1,3)
Meeting with 1st Intaker	1st Intaker	Exponential (30)
Meeting with 2nd Intaker	2nd Intaker	Exponential (20)
Type out Conversation	1st Intaker	Beta (5,10)
Complete File with 2nd Info	Secretarial Office Worker	Uniform (1,3)
Complete File with 1st Info	1st Intaker	Uniform (1,3)
Determine Treatment	Treatment Team	Beta (5,15)

After Store and Print notice the decision is made whether the patient is known or not. This can be modeled using a Decide block. A patient file is created when the patient is not known, otherwise this activity is skipped. After Close Case the entity waits until Wednesday. The entity waits in a Hold block until the condition: CalDayOfWeek(tnow)==4 is true (DayOfWeek 4 = Wednesday). When it is Wednesday the entity continues. The intakers are assigned and the assignment is stored. When it is necessary the medical file is asked for and the patient file is updated. Concurrently the cards are handed out. This is modeled using a multi choice construct. In Arena a Branch block is used, that always sends an entity to hand out cards and one entity to either the second or the third branch. After that the meetings with the first and second intakers are planned and executed, the conversations are typed out and the files are completed. The entity has to wait again until the first Wednesday. This time it waits in Hold for Wednesday 2. After that the treatment is determined. This is done by the treatment team, consisting of a first intaker, a second intaker, a psychiatrist and a team leader. This can be modelled in Arena by allocating the four resources to the process “Determine Treatment”.

In the Resource module in Arena can be specified what resources there are in a process and what the specification of every single resource is.

9. Conclusions

The following table summarizes the results of the translation of the workflow patterns into Arena. If there is a building block to model a workflow pattern and the pattern can be modelled straightforward, the score will be a +. If the pattern is not directly supported, or an increase in complexity will occur when expanding the model, the score will be +/-. If it is not possible to model the workflow pattern in Arena, the score will be -.

Pattern	**Score**	**Motivation**
1. Sequence	+	Directly supported
2. Parallel split	+	Directly supported
3. Synchronization	+	Directly supported
4. Exlcusive Choice	+	Directly supported
5. Simple Merge	+	Directly supported
6. Multi Choice	+	Directly supported
7. Synchronizing Merge	+/-	The more parallel branches, the more complex the model gets.
8. Multiple Merge	+	Directly supported
9. Discriminator	-	Not supported
9a. N-out-of-M	-	Not supported, only constructs with N>1 and M<2N-1
10. Arbitrary cycles	+	Directly supported
11. Implicit Termination	-	Not supported
12. Multiple instances without synchronization	+	Directly supported
13. Multiple instances with a priori design time knowledge	+	Directly supported
14. Multiple instances with a priori runtime knowledge	+/-	Only when the stated assumption holds
15. Multiple instances without a priori runtime knowledge	+/-	Only when the stated assumption holds
16. Deferred choice	-	
17. Interleaved parallel routing	+/-	The more parallel branches, the more complex the model gets.
18. Milestone	+	Directly supported
19. Cancel activity	+	Directly supported
20. Cancel Case	-	Not supported

Because of the complexity of Arena, it depends very much on the situation if it is possible or not to model a certain case. Most of the important and normally necessary patterns are supported by Arena (pattern 1 to 5). This makes it possible to model a lot of different cases, using Arena.

Arena is not a software package that is specially developed to model workflows. That is why not all workflows are supported.

References

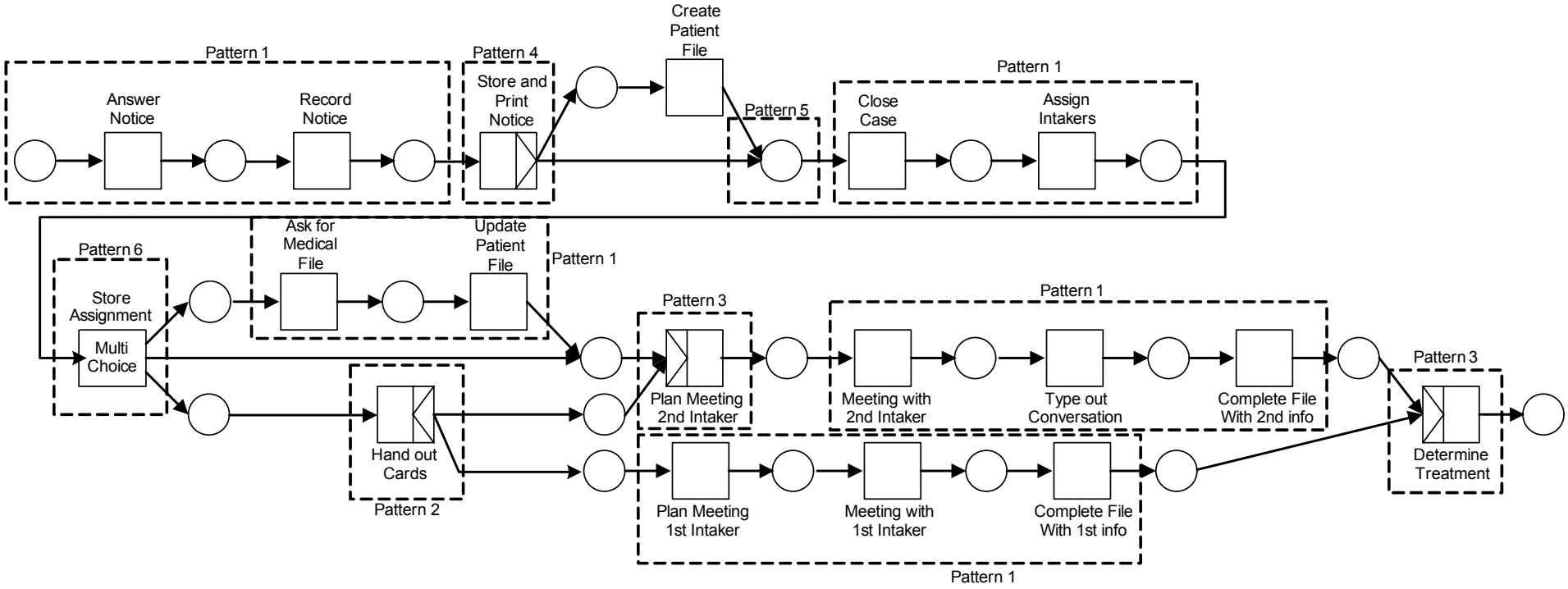
[1] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski and A.P. Barros. Workflow Patterns. *Distributed and Parallel Databases,* 14(3), pages 5-51, July 2003

[2] www.workflowpatterns.com

[3] W. David Kelton, Randall P. Sadowski, David T. Sturrock. Simulation with Arena, Third edition, 2004

[4] H.A. Reijers. Design and Control of Workflow Processes, Proefschrift, 2002

Appendix 1 Workflow Grote Beek using Petri net notation



Appendix 2 Grote Beek in Arena