

# Operon Technical Design

## *Synopsis*

This document details the design of Operon, a powerful, lightweight workflow engine that is based on the expressive Petri-net notation.

*Author(s)*      ***Chung Hua***

*Published*      ***25 April 2004***

*Version*      ***0.1 Draft***

# Table of Contents

<b>1</b>	<b>DOCUMENT MANAGEMENT</b>	<b>4</b>
1.1	Version History .....	4
1.2	Distribution .....	4
1.3	References .....	4
1.4	Glossary .....	4
<b>2</b>	<b>INTRODUCTION</b>	<b>5</b>
<b>3</b>	<b>WORKFLOW</b>	<b>6</b>
<b>4</b>	<b>PETRI-NET</b>	<b>7</b>
4.1	Overview .....	7
4.2	Objects in Petri-net .....	7
4.3	Petri-net rules .....	7
4.4	Subnet .....	8
<b>5</b>	<b>WF-NETS</b>	<b>9</b>
5.1	Overview .....	9
5.2	Transition Types .....	9
5.3	Triggers .....	11
5.4	Routings .....	12
5.5	Sample WF-net .....	13
<b>6</b>	<b>OPERON</b>	<b>15</b>
<b>6.1</b>	<b>Design Considerations .....</b>	<b>15</b>
6.1.1	Scalability .....	15
6.1.2	Extensibility .....	15
6.1.3	Clustering .....	15
6.1.4	Transaction .....	15
6.1.5	Asynchronous Processing .....	15
6.1.6	Standard .....	15
<b>6.2</b>	<b>Physical Architecture .....</b>	<b>16</b>
<b>6.3</b>	<b>UML Model .....</b>	<b>16</b>
6.3.1	Domain Objects .....	16
6.3.2	Business Objects .....	17
6.3.3	Application Customisation Interfaces .....	17
<b>6.4</b>	<b>Workflow Object Statuses .....</b>	<b>18</b>
6.4.1	Case Statuses .....	18
6.4.2	WorkItem Statuses .....	19
6.4.3	Token Statuses .....	20
<b>6.5</b>	<b>Operon Data Model Partition .....</b>	<b>22</b>
6.5.1	CONFIG_REPOSITORY .....	22
6.5.2	OPERON_CASE .....	23

6.5.3	TTL_SCHEDULER .....	23
6.5.4	OPERON_TASK .....	25
6.5.5	TIME_TRIGGER_SCHEDULER.....	26
6.5.6	OPERON_EVENT_AUDIT.....	26
6.5.7	OPERON_TOKEN .....	28
6.5.8	OPERON_TOKEN_PLACE_REF .....	28
6.5.9	OPERON_TOKEN_ENABLED_TASK.....	29
<b>6.6</b>	<b>Uses Cases And Sequence Diagrams .....</b>	<b>30</b>
6.6.1	Load Nets.....	31
6.6.2	Open Case.....	31
6.6.3	Suspend Case .....	32
6.6.4	Resume Case .....	32
6.6.5	Cancel Case .....	33
6.6.6	Start WorkItem.....	33
6.6.7	Finish Activity.....	33
<b>7</b>	<b>DEVELOPERS GUIDE .....</b>	<b>35</b>
<b>7.1</b>	<b>Reusable Frameworks and Components .....</b>	<b>35</b>
<b>7.2</b>	<b>Operon XML Configuration File.....</b>	<b>35</b>

# 1 Document Management

## 1.1 Version History

Version	Date	Author	Details
0.1	24 Feb 2004	C Hua	Initial draft

## 1.2 Distribution

Person	Company	Role

## 1.3 References

The following list references documents that were used to help write this document.

Reference	Source
[1]	The Application of Petri Nets to Workflow Management by W.M.P. Wil van der Aalst, Department of Mathematics and Computing Science, Eindhoven University of Technology

## 1.4 Glossary

Term	Definition
JMS	
MDB	
Petri-net	
Workflow	
Activity	A workItem in IN_PROGRESS status
Explicit Timer	The job is fired exactly at the time specified
Implicit Timer	A timer that runs at regular intervals and picks up any job that is due to be fired but has not fired yet. This mean the job may not be fired at the exact time but will either be on the specified time or later.

## 2 Introduction

Operon is a workflow engine that uses the Petri-net based notation to define the workflow. This document describes the philosophy behind Operon by describing what is called a workflow and how workflows can be express as a Petri-net.

The latter of the document details the design of the Operon workflow engine.

### 3 Workflow

Workflows are **case**-based, i.e., every piece of work is executed for a specific case. Examples of cases are a mortgage, an insurance claim, a tax declaration, an order, or a request for information. Cases are often generated by an external customer. However, it is also possible that a case is generated by another department within the same organization (internal customer).

The goal of workflow management is to handle cases as efficiently and effectively as possible. A **workflow process** is designed to handle similar cases.

Cases are handled by executing **tasks** in a specific order. The workflow process definition specifies which tasks need to be executed and in what order. Alternative terms for workflow process definition are: 'procedure', 'flow diagram' and 'routing definition'. Since tasks are executed in a specific order, it is useful to identify conditions which correspond to causal dependencies between tasks. A condition holds or does not hold (true or false). Each task has pre- and postconditions: the preconditions should hold before the task is executed, and the postconditions should hold after execution of the task.

Many cases can be handled by following the same workflowprocess definition. As a result, the same task has to be executed for many cases. A task which needs to be executed for a specific case is called a **work item**. An example of a work item is: execute task 'send refund form to customer' for case 'complaint sent by customer Baker'.

Most work items are executed by a **resource**. A resource is either a machine (e.g. a printer or a fax) or a person (participant, worker, employee). In most offices the resources are mainly human. However, because workflow management is not restricted to offices, we prefer the term resource. Resources are allowed to deal with specific work items. To facilitate the allocation of work items to resources, resources are grouped into classes. A resource class is a group of resources with similar characteristics. There may be many resources in the same class and a resource may be a member of multiple resource classes. If a resource class is based on the capabilities (i.e. functional requirements) of its members, it is called a role. If the classification is based on the structure of the organization, such a resource class is called an organizational unit (e.g. team, branch or department).

A work item which is being executed by a specific resource is called an **activity**.

If we take a photograph of a workflow, we see cases, work items and activities. Work items link cases and tasks.

There can be two basic types of workflow:

Activity based - means that the processes, the workflows, are made of activities to be completed in order to get something done.

Entity based - means that the focus is set on a given document and the states it has to go through in order to be completed.

## 4 Petri-net

### 4.1 Overview

In order to implement a workflow system it is first necessary to find a suitable means of designing and modeling a workflow process. For this we can use the work done by Carl Adam Petri who was the first to formulate a general theory for discrete parallel systems which gave birth to what are now known as Petri Nets.

Petri Nets is a formal and graphical language which is appropriate for modelling systems with concurrency and resource sharing. It is a generalisation of automata theory such that the concept of concurrently occurring events can be expressed.

Petri Nets have become so popular and widespread that there is a Platform Independent PetriNet Editor (PIPE) available, and it even has its own Petri Net Markup Language (PNML).

### 4.2 Objects in Petri-net

Places	These are inactive and are analogous to inboxes in an office-based system. They are shown as circles in a Petri Net diagram. Each Petri Net has one start place called a <b>source</b> and one end place called a <b>sink</b> , but any number of intermediate places.
Transition	These are active and represent <b>tasks</b> to be performed. They are shown as rectangles in a Petri Net diagram.
Arcs	Each of these joins a single <b>Place</b> to a single <b>Transition</b> . They are shown as connecting lines in a Petri Net diagram. An <b>inward arc</b> goes from a <b>Place</b> to a <b>Transition</b> and an <b>outward arc</b> goes from a <b>Transition</b> to a <b>Place</b> .
Tokens	These represent the current state of a workflow process. They are shown as <b>black dots</b> within <b>Places</b> in a Petri Net diagram. A <b>place</b> can hold zero or more tokens at any moment in time.

### 4.3 Petri-net rules

1	<b>Places</b> do nothing but hold <b>Tokens</b> representing the state of the process. A <b>Place</b> can hold zero or more <b>Tokens</b> at any moment in time.
2	An <b>Arc</b> connects a <b>Place</b> to a <b>Transition</b> . - Place P is called an <b>input place</b> of transition T if there exists a directed arc from P to T. - Place P is called an <b>output place</b> of transition T if there exists a directed arc from T to P.
3	When an <b>enabled transition</b> is fired it moves tokens from all its <b>input places</b> to all its <b>output places</b> . - Transition T is said to be <b>enabled</b> if each input place P of T contains at least one token. - How an enabled transition is actually fired depends on the type of <b>trigger</b> . - When transition T is fired it <b>consumes</b> one <b>token</b> from each <b>input place</b> P of T and <b>produces</b>

	<p>one <b>token</b> in each <b>output place</b> P of T.</p> <p>To represent how many tokens a transition consumes and produces we use <b>weighted</b> arcs, where a weight is an integer expression. Then, when a transition fires it consumes a number of tokens equal to the weight of the edge for each input and produces a number of tokens of equal weight of the edge for each output.</p>
4	When an <b>enabled transition</b> is fired it moves tokens from all its <b>input places</b> to all its <b>output places</b>
5	Each workflow process has a single start place ( <b>source</b> ). It must have at least one inward arc going into a transition. It may have an outward arc coming from a transition in order to restart the process.
6	Each workflow process has a single end place ( <b>sink</b> ). It must have at least one outward arc coming from a transition (it may have more than one), but it cannot have any inward arcs going into transitions.

## 4.4

**Subnet**

To allow multiple hierarchy sub processes the concept of subnets are used. A subnet is a Petri-net inside another Petri-net and is linked via a parent child relationship where the child is the subnet of the parent.

A subnet has all the attributes of a Petri-net. The only difference is the subnet **does** have a **Source Place** or A **Sink Place**. Instead it has an **InReference Place** and an **OutReference Place**.

An **InReference Place** is a reference from the parent net intermediate place into the starting place of the subnet.

An **OutReference Place** is an end place of the subnet that has a reference linking to the parent net intermediate place.

There must be one **InReference** and one **OutReference** Place in the Subnet.



## 5 WF-Nets

### 5.1 Overview

In order to map workflows to Petri-Nets we extend the classical Petri-Nets notation to WF-nets established by Wil van der Aalst (TU Eindhoven).

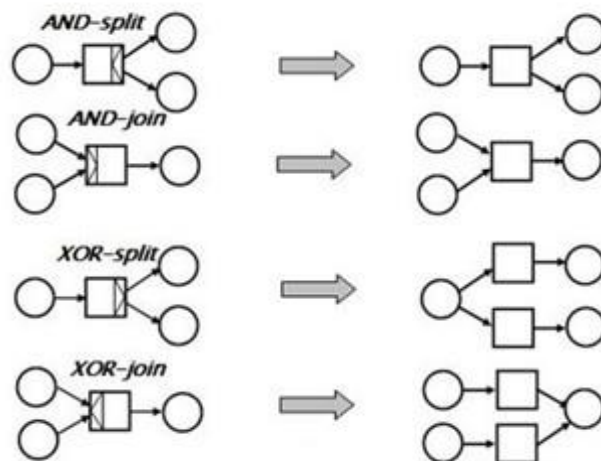
A WF-Net is an extension of the Petri-net and it is supposed to have a very regular structure: It must contain exactly one place with no incoming arcs (the input place) and exactly one place with no outgoing arcs (the output place). Moreover, the net graph (extended by one "virtual" transition with an arc from the output place and an arc to the input place) must be strongly connected, i. e. from each node there exists a directed path to any other node.

### 5.2 Transition Types

**Transitions** are analogues to **Task**, Transitions are used in Petri-net terms and Tasks are used in Workflow terms. These terms are interchangeable.

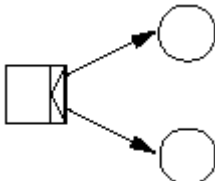
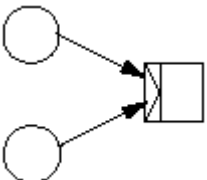
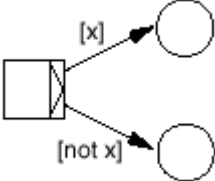
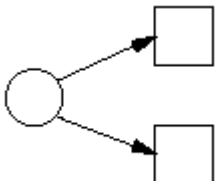
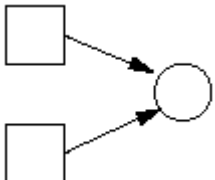
Four special transition types are added to express branching situations in a more compact way: AND-split, AND-join, XOR-split and XOR-join, each of them being associated with a graphical symbol.

Conceptionally, these special transitions are nothing more than shortcuts ("syntactic sugar") for an underlying equivalent standard Petri Net, such that the formal nature and executability of the Petri net calculus is preserved (see diagram below).



AND Split

An example of **parallel routing** where several tasks are performed in parallel or in no particular order. It is modeled by a transition with one input place and two or more output places. When fired the transition will create tokens in all output places.

	
<p>AND Join</p> 	<p>A transition with two or more input places and one output place. This will only be enabled once there is a token in all of the input places, which would be after each parallel thread of execution has finished.</p>
<p>Explicit OR Split</p> 	<p>An example of <b>conditional routing</b> where the <b>decision is made as early as possible</b>. It is modelled by attaching conditions or <b>guards</b> to the arcs going out of a transition.</p> <p><b>Guard</b> - An expression attached to an arc, shown in brackets, that evaluates to true or false. Tokens can only travel over arcs when their guard evaluates to true. The expression will typically involve the case attributes.</p>
<p>Implicit OR Split</p> 	<p>An example of conditional routing where the <b>decision is made as late as possible</b>. Implicit or-splits are modelled as two arcs going from the same place but to different transitions. That way, the transition that happens to fire first (which depends on the transition trigger) will get the token. Once the token is gone, the others are no longer enabled and thus cannot be fired.</p>
<p>OR Join (Explicit and Implicit)</p> 	<p>Is simply a place that serves as the output place of two different transitions. That way, the next transition after the or-join place will be enabled when either of the two conditional threads are done.</p>

## 5.3





## Triggers

The time the transition is enabled and the time it fires are different. The thing that causes an enabled transition to fire is called a trigger.

Triggers are added to the standard Petri net notation in order to represent different kinds of dependency between a task of the workflow process and its operative environment.

WF-nets allow assigns exactly one trigger type to every transition. The trigger type is graphically represented by a small, self-explaining icon near the associated transition symbol.

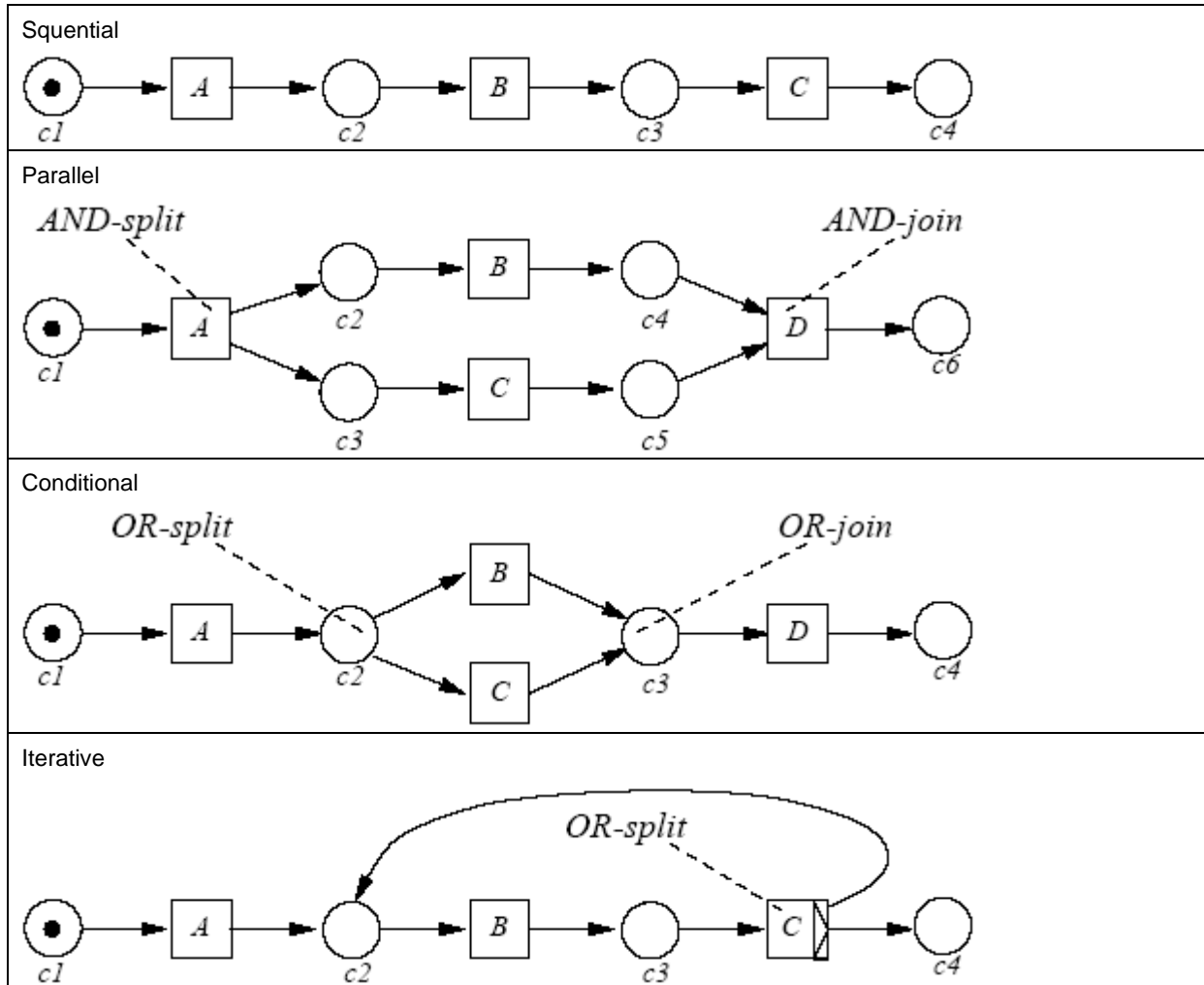
There are four different types of trigger:

	Automatic	<p>A task is triggered the moment it is enabled. This kind of triggering is used for tasks which are executed by an application which does not require human interaction.</p> <p>Once this triggers is pulled the transition will execute and end automatically.</p>
	User (Manual Trigger)	<p>A task is triggered by a human participant, i.e., a user selects an enabled task instance to be executed. In a workflow management system each user has a so-called 'in-basket'. This in-basket contains task instances (WorkItems) that are enabled and may be executed by the user. By selecting and completing a WorkItem the corresponding task instance is triggered and the workflow case is advanced to the next stage in the process.</p> <p>Once the user manually pulls the trigger the Transition will start but requires the user to manually tell the system when the Transition ends.</p>
	Time (Semi-Automatic Trigger)	<p>An enabled task instance is triggered by a clock, i.e., the task is executed at a predefined time. For example, the task 'remove document' is triggered if a case is trapped in a specific state for more than 15 hours.</p> <p>This should be one of the options in an implicit OR split.</p> <p>Once this triggers is pulled the Transition will execute and end automatically.</p>
	Message (Semi-Automatic Trigger)	<p>An external event (i.e. a message) triggers an enabled task instance. Examples of messages are telephone-calls, fax messages, e-mails or EDI messages.</p> <p>Each of these external events will probably require some action within an application task so that the workflow system is made aware that the event has taken place.</p> <p>Once this Trigger is pulled the Transition will execute and end automatically.</p>

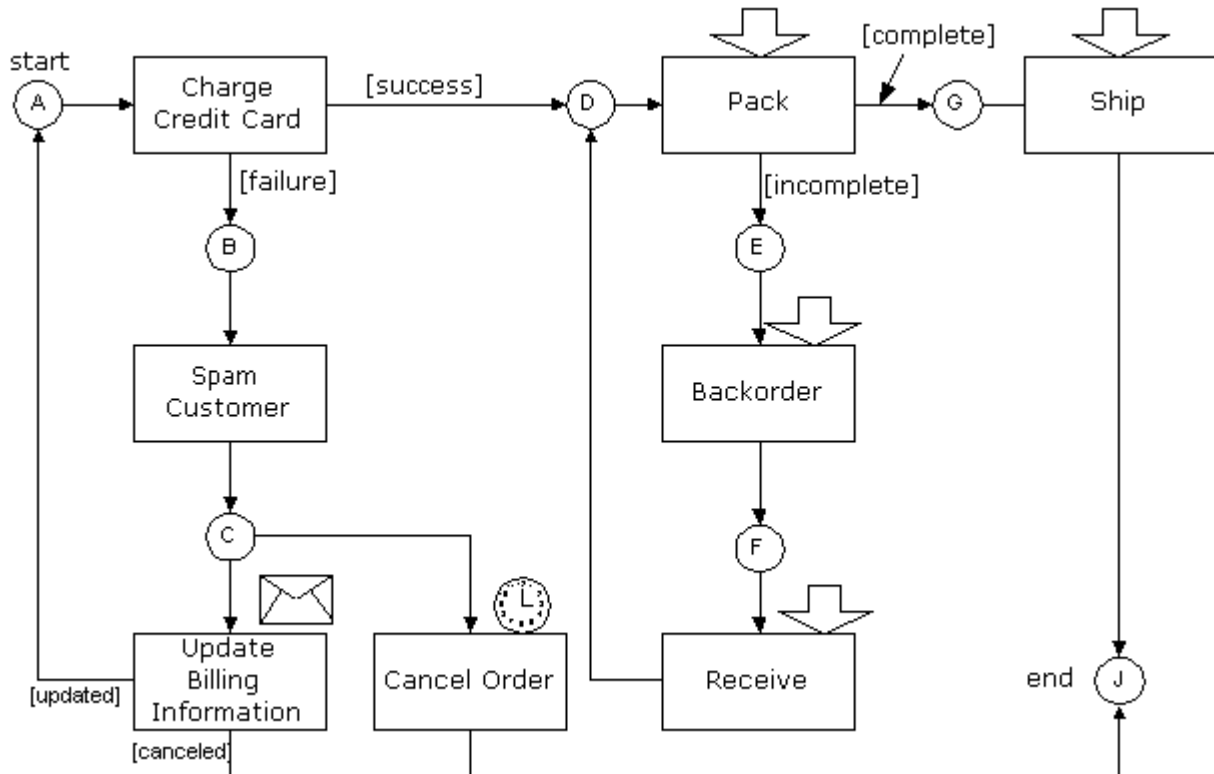
## 5.4

## Routings

The route between the start place and the end place within a workflow process using the previously defined special transitions as control flow elements can take several forms. These are:



## 5.5

**Sample WF-net**

The explanation of this diagram is as follows:

The circles are called **places** and represent the **inboxes** in the office-metaphor.

The rectangles are called **transitions** and represent the **tasks** to be performed.

**Places** are inactive. All the **places** do is hold **tokens** representing the state of the process. If, for example, there's a **token** in **place D** above, then that means we're ready to pack the order.

**Transitions** are active. They move **tokens** from their **input places** (the places that have an **arc** pointing into the transition) to their **output places** (the places you get to by following the **arcs** going out of the **transition**). When this happens the **transition** is said to **fire**.

**Transitions** can only **fire** when there is at least one **token** in each **input place**. When that is the case, the **transition** is **enabled**. That the transition is **enabled** means it is able to fire. It will fire when the conditions of its trigger are satisfied.

When the workflow is started, a **token** is placed in the start **place** (A in the example). This enables the automatic **transition** 'Charge Credit Card'.

The transition fires with a success or a failure. If it was successful, it **produces a token** in **place** D. If there was a failure, it **produces a token** in **place** B. Thus, the outcome of the attempt at charging the credit card governs the further routing of the process. The rule is that firing a transition **consumes** one **token** from each of its **input places**, and places a **token** on each of its **output places** for which the **guard** is true. The **guard** is a predicate, in this case the [success] and [failure] on the **arcs** going out of 'Charge Credit Card'. **Guards** are what enables us to do conditional routing. The 'Charge Credit Card' **transition** acts as an **explicit or-split**, because it chooses either one route or the other.

The other form of **conditional routing**, which is the **implicit or-split**, is what chooses between the **transitions** 'Update Billing Information' and 'Cancel Order'. Since there's only one **token** in **place** C, only one of the two **transitions** can have it. But, contrary to the **explicit or-split**, where the decision is explicitly made as soon as 'Charge Credit Card' finishes, the choice between 'Update Billing Information' and 'Cancel Order' is made as late as possible.

Both transitions will be enabled when there's a **token** in **place** C (i.e. when the spam has been sent). If the user updates his billing information before the timed 'Cancel Order' transition times out, 'Cancel Order' is never fired. And vice versa: If the order is cancelled (which will probably involve spamming the user again to let him know that his order was cancelled), then he won't be able to update his billing information and will have to enter a new order. Thus, the choice is made implicitly, based on the timing.

The **guard** will generally depend on **case attributes**. The 'Charge Credit Card' transition above will set a **case attribute** to either 'success' or 'failure', and the guard will check this value to determine its result. **Case attributes** can hold more complex values than simple yes/no values, but the guard must always be either true or false.

What is missing from this diagram is the process which initiates a new workflow instance (or 'case') and puts a token in the start place. In the above example the case initiator would be 'Take Customer Order'.

## 6 Operon

*OPERON originates from the field of genetics a group of adjacent genes functioning as a unit under the control of another gene (the operator gene).*

In computing terms a group of adjacent genes could be a group of components all working together under the control of a workflow engine (the operator). Here, the workflow engine controls the group of components and decides who does what and when depending on the defined business process.

The design of Operon is based on Wfnets Ref [1] (Dr Van Der Aalst paper on The Application of Petri Nets to Workflow Management). It is a recommendation to read this paper first before continuing.

### 6.1 Design Considerations

#### 6.1.1 **Scalability**

The workflow engine must be able to support small to large scale enterprise business processes. Therefore it is appropriate to implement the workflow engine as a J2EE application.

#### 6.1.2 **Extensibility**

Applications using the Operon framework are required to implement two interfaces:

ResourceManager interface allows the Operon framework to interface with the application access controls.

Action interfaces allows Operon to execute application specific business logics encapsulated within the Action classes.

#### 6.1.3 **Clustering**

Since Operon supports J2EE, it can be deployed onto any J2EE server, its up to the server to handle the clustering of the application.

#### 6.1.4 **Transaction**

The Spring framework Transactions will be used to support transactions

#### 6.1.5 **Asynchronous Processing**

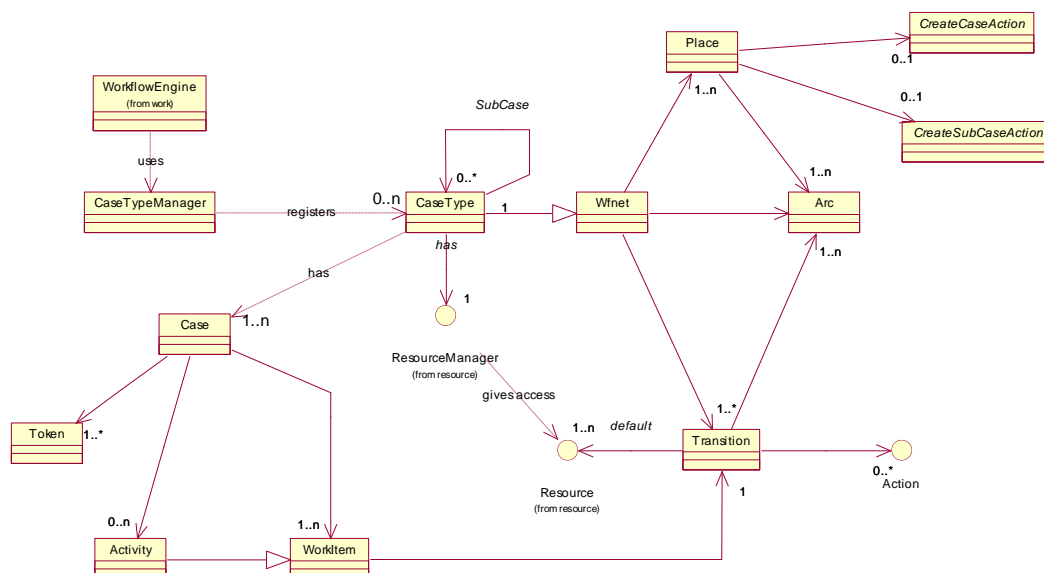
Quartz schedulers will be used to support asynchronous processing.

#### 6.1.6 **Standard**

In order to use existing Petri-net simulation tools to verify the Operon Wfnet the PNML Petri-net language will be used. The advantage of using PNML is that it allows extensions for specific tools. Operon will use Operonml which is an extension of the PNML language that is specific for Operon.

## 6.2 Physical Architecture

### 6.3 UML Model



### 6.3.1 *Domain Objects*

Name	Description
Wfnet	An abstract class used to represent a static WF-net
Token	Tokens are used to represent the current state of an instance of a Wfnet.
Place	<p>A Place holds Tokens within a WF-net. There are four types of Places within the Wfnet:</p> <ol style="list-style-type: none"> <li>1. Sink – a starting place, there must be one starting place</li> <li>2. Source – an end place, there must be one end place</li> <li>3. Intermediate – intermediate places, there can be any number intermediate places</li> <li>4. InRef – If the net is a subnet, there must be one inRef place which links to a parent intermediate place.</li> <li>5. OutRef – an out reference place of a subnet, there must be one inRef place which links to a parent intermediate place (or sink place)</li> </ol>
Transitions	<p>Transitions are analogues to Tasks in the Workflow Context</p> <p>Transitions are used to move tokens from one place to another.</p> <p>An enabled Transition contains a Token from Input an input place.</p> <p>An enabled Transition when fires consumes a Token(s) from its input Place and produces A Token(s) to its output Place.</p> <p>A Transition is associated with many Actions and Resources.</p>
Action	Is an interface, implementations of this interface can be registered with a Transition,



	when the Transition is fired these Actions are automatically executed. The calling application can encapsulate its business logics within these Action classes.
Resource	The Resource is an Interface. Each Transition is associated with at least one Resource, the Resource determines who can fire the transaction. The calling application must implement its own version of the interface.
Arc	An Arc connects a Place and Transition together within the WF-net.
CaseType	An extension of the WF-net.
Case	An instance of a CaseType that is associated with WorkItems and Activities.
WorkItem	Is an ENABLED transition i.e. the Transition can be fired.
Activity	Is a fired Transition i.e. the Transition is in PROGRESS

### 6.3.2 ***Business Objects***

Name	Description
WorkflowEngine	This is controls the flow of the Wfnet
CaseTypeManager	Responsible for registering all different type of cases (WFnet)

### 6.3.3 ***Application Customisation Interfaces***

In order to use the Operon framework the calling application must implement the following interfaces and register the implementations within the Operon.xml configuration file.

Name	Description
ResourceManager	Different companies have different access and authentication policies therefore it is not in Operon's interest to implement its own version. For Operon to use the calling application access control the calling application will have to implement a ResourceManager interface and register the interface within the Operon framework.  The ResourceManager interface allows Operon to find Resources ids registered in the Operon.xml config files and registered those resources with the newly created WorkItems. The ResourceManager interface is also used to check if a Resource is allowed to execute or cancel a WorkItem or an Activity respectively.
Action	Whenever a Task is executed it becomes an Activity, all Actions registered within the Transition that is associated with the Activity will be executed.  Application specific business rules and process logics are encapsulated within the Action classes.
CreateCaseAction	This is an abstract class. All Source Place will be associated with a CreateCaseAction.  Apart from encapsulating business logics the CreateCaseAction class is useful for external applications to link the new created Case with other external entities.

CreateSubCasesAction	This is an abstract class. All inref Places will be associated with a CreateSubCasesAction. The CreateSubCases allows multiple sub-cases to be created in one go. Apart from encapsulating business logics, sometimes the execution of the parent Case may result in N sub-cases. The class is also useful to link external entities to the newly created sub-cases.
----------------------	--

## 6.4

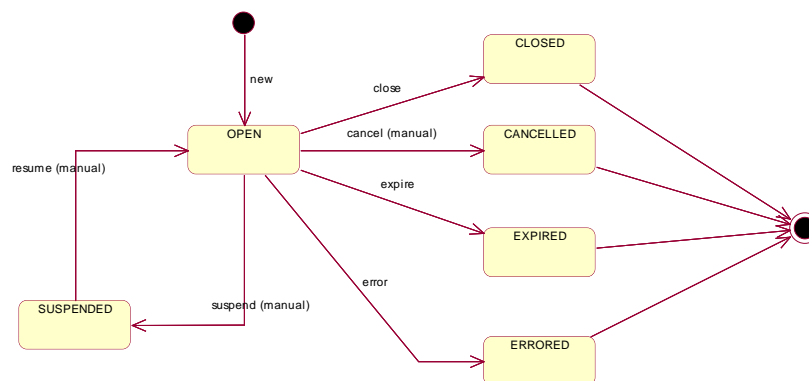
**Workflow Object Statuses**

Within the Operon Workflow key two entities have statuses these are a Case and its associated WorkItems.

## 6.4.1

**Case Statuses**

The diagram below shows the possible statuses for a Case.



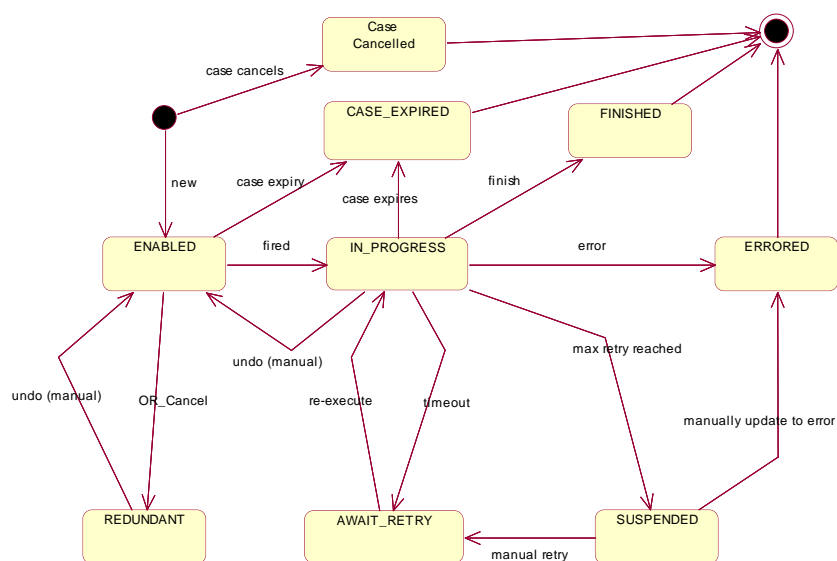
Status	Description
OPEN	When a new case is created it will be in an OPEN status
CLOSED	When the Case ends
CANCELLED	Cancelled can occur when a user/application manually cancels the case. Once cancelled the Case cannot be re-OPEN, a new case will have to be created.
SUSPENDED	Cases can be suspended and resume when required. For example if an external system is down and Operon cannot communicate with the external system for a particular Case it is advisable to SUSPEND the Case until the external system is in operational again. This is triggered manually by the application.  If the Case is SUSPENDED then its existing WorkItems will not be processed. All existing automatic and semi-automatic triggered Activities will be allowed to finish its execution.
ERRORED	This can happen during the execution of a WorkItem.  There was some sort of runtime system error with an external resource that the WorkItem is trying to communicate with and after the max retries count is reached it will go into ERROR. This normally occurs in RUNTIME EXCEPTIONS.  A piece of business data that is corrupted and cannot be process can cause the system to go into ERRORED. This normally occurs in APPLICATION EXEPTIONS.

	Once ERRORED the Case cannot be re-OPEN. A new Case have to be created.
EXPIRED	If the Case is given a time limit and the limit has been reached it will EXPIRED. Once EXPIRED the Case cannot be re-OPEN and a new case has to be created instead.

## 6.4.2

**WorkItem Statuses**

The diagram below shows the possible statuses for a WorkItem.



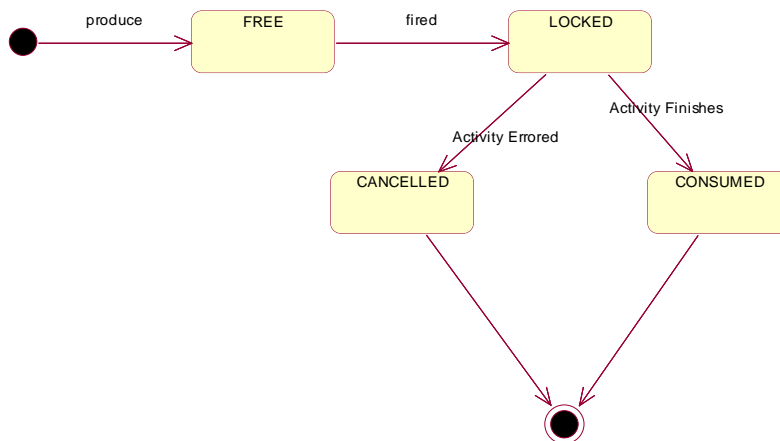
Status	Description
ENABLED	The WorkItem and is ready to be fired
IN_PROGRESS	The WorkItem being executed (IN_PROGRESS) is also called an <b>Activity</b> . For a manual trigger the user can <i>undo</i> the Activity back to a WorkItem (from IN_PROGRESS back to ENABLED).
FINISHED	When the WorkItem finishes executing.
REDUNDANT	Cancels the WorkItem. This can happen during an implicit OR when a Place with a Token has two output Arcs to two different enabled Transitions (WorkItems) and only one of them can be executed depending on the one that is executed first. The first one to execute will make the other one REDUNDANT.
ERRORED	This can happen during an automatic execution of an Activity.  A piece of business data that is corrupted and cannot be process can cause the system to go into ERRORED. This normally occurs in APPLICATION EXEPTIONS.  For a Message Triggered execution of a WorkItem, the calling application can

	<p>manually instruct Operon to put the Activity into ERRORED.</p> <p>For manual Trigger, the user can choose to put the WorkItem to ERRORED if they wish.</p> <p>Once ERRORED the whole Case will also go into ERRORED.</p>
SUSPENDED	<p>For auto/semi automatic triggered transitions, the Activity can go into SUSPENDED when it has timeout and the maximum retry is reached.</p> <p>The SUSPENDED state allows the user to pause the workflow and fix external problems such as a faulty fax machine or restarting an email server etc.</p> <p>Once in SUSPENDED state the user has two options:</p> <ol style="list-style-type: none"> <li>1. Manually put the Activity back into AWAIT_RETRY status so that it can be re-excuted.</li> <li>2. Manually put the Activity to ERRORED and stating the detail of the error. This is only used if the problem cannot be fixed.</li> </ol>
AWAIT_RETRY	If an Activity timeout it will put into this state so that it can be re-executed (retry).
CASE_EXPIRED	If the Root Case expires then all WorkItems and Activities will move into CASE_EXPIRED
CASE_CANCELLED	When the Root Case is manually cancelled all Manual and Message Triggered WorkItems and Activities will be cancelled.

## 6.4.3

**Token Statues**

The diagram below shows the possible states of a Token.

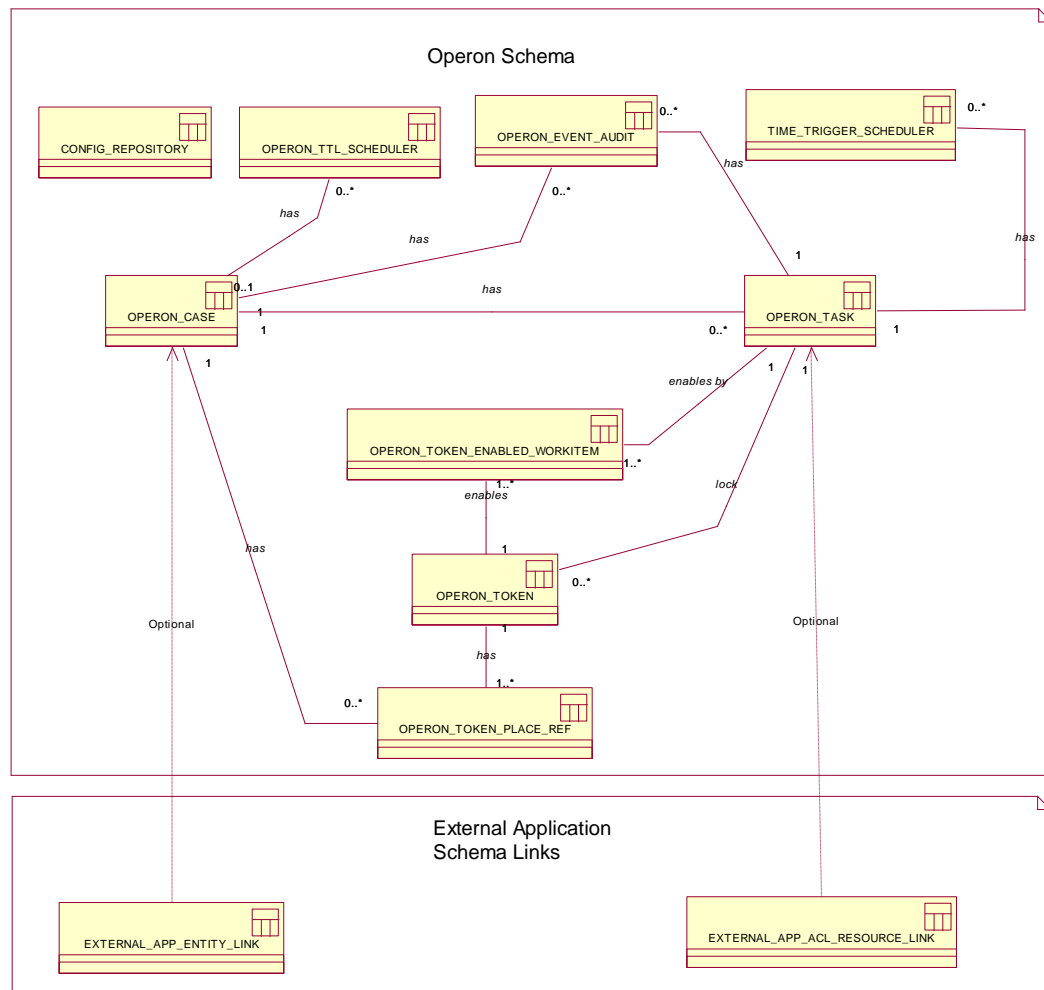


Status	Description
FREE	When the Token is first produced (either by a finished Activity) or at the start of the Case.
LOCKED	When the Token is fired it is locked by the Activity

CONSUMED	When an Activity finishes executing in Consumes the lock Token
CANCELLED	If an Activity goes into Error the Token will be Cancelled so that is no longer available.

## 6.5 Operon Data Model Partition

This section contains Operon data model.



**Note:** The External Application Schema links only serve as an example of how external applications can link to the Operon Schema. The table name within the external schema can be anything.

### 6.5.1 CONFIG\_REPOSITORY

This is optional if the user wishes to store XML Config files in the database.

Column	Constraint	Type	Description
SERVER_ID	PK	VARCHAR(50)	The server the JVM is running in

APPLICATION_NAME	PK	VARHAR(50)	The application name
FILE_NAME	PK	VARCHAR(50)	The filename
CONTENT	NOT NULL	TEXT or CLOB	Stores the XML content
CREATED_DATE	NOT NULL	DATE	The time stamp of when this was created
UPDATED_DATE	NOT NULL	DATE	The time stamp of when this column was updated

### 6.5.2 **OPERON\_CASE**

This maps to the domain model Case. A Case is an instance of a Case Type.

Column	Constraint	Type	Description
CASE_ID	PK	NUMBER	A unique key to identify a Case
PARENT_CASE_ID	FK	NUMBER	The parent Case if this Case is a subnet.
ROOT_CASE_ID	FK	NUMBER	The root parent Case if this is a subnet.
CASE_TYPE_REF	NOT_NULL	VARCHAR(50)	A reference to the name of the Wfnet (net id) defined in the Operon configuration file. In the case of a Subnet this references the <Page id> tag.
CASE_STATUS	NOT NULL	VARCHAR(9)	The possible options are: OPEN, CLOSED, SUSPENDED, EXPIRED, and ERRORED. See Case Statuses for details
EXPIRY_DATE	ALLOW NULL	DATE+TIME	The date when this case expires. This is computed from the current sys date + the time to live in the XML config.
LOCK_VERSION	NOT NULL	NUMBER	Default is 0
CREATED_DATE	NOT NULL	DATE	The time stamp when this case was created.
UPDATED_DATE	NOT_NULL	DATE	The time stamp of when this column was updated

### 6.5.3 **TTL\_SCHEDULER**

(TIME TO LIVE) schedulers - This is only used if the OPERON\_CASE.EXPIRY\_DATE is not null. If the OPERON\_CASE has associated TTL\_SCHEDULERS then this implies we are using implicit timers otherwise its explicit timers.

1. Explicit - the Scheduler will execute straight away once expired.
2. Implicit - the Scheduler will use one of the registered schedulers and execute at the next round of execution.

Column	Constraint	Type	Description
CASE_ID	FK	NUMBER	A Foreign key to the associated Case
SCHEDULER_REF	NO_NULL	VARCHAR(50)	Points to a registered scheduler within the XML config

			file.
CRON_EXP	NOT_NULL	VARCHAR(80)	The Crontab expression. This is going to be used by the scheduler as the reference id to execute the WorkItem. E.g. If the registered scheduler with CronExp "0 0/2 * * * ?" then every time it executes it will search for all CRON_EXP that matches it and executes the WorkItem. This means only one scheduler for that particular time is created globally.
CREATED_DATE	NOT NULL	DATE	The time stamp when this case was created.
UPDATED_DATE	NOT_NULL	DATE	The time stamp of when this column was updated



#### 6.5.4 **OPERON\_TASK**

This maps to the domain model WorkItem and Activity.

Note: A WorkItem is a Task is ENABLED status and an Activity is Task IN\_PROGRESS.

Column	Constraint	Type	Description
TASK_ID	PK	NUMBER	A unique key to identify the WorkItem.
CASE_ID	FK	NUMBER	The associated Case
TASK_STATUS	NOT NULL	VARCHAR(11)	Possible options are: 1. ENABLED – in Wfnet terms the Transition is enabled and is ready to be fired i.e. a WorkItem 2. IN_PROGRESS – in Wfnet terms the Transition has started (fired) and is now called an Activity. 3. FINISHED – in Wfnet terms the Activity has finished executing. 4. REDUNDANT – For an XOR Transition if another Transition is executed before this one then it will make all the other Transitions in the XOR branch REDUNDANT. 5. ERRORED – see Task Statuses 6. SUSPENDED – see Task Statuses 7. AWAIT_RETRY – see Task Satuses
TRIGGER_TIME	ALLOW NULL	DATE+TIME	If the TRANSITION trigger-type=TIME then this is the date and time when the transition will trigger.
WFNET_TRANSITION_REF	NOT NULL	VARCHAR(50)	The reference to the Wfnet Transition defined in the Operon configuration file.
START_AT_STARTUP	NOT NULL	VARCHAR(1)	Default is N otherwise Y  This is use for Automatic and Time Triggered Transitions after downtime periods such as power failures.  When the application restarts all Automatic and Time Triggered ENABLED Workitems will be re-started.  In the case of Time Triggered Transitions that has passed its due Triggered Time and the WorkItem is still Enabled then it will be automatically started.
IN_PROGRESS_TIMEOUT_DATE	ALLOW NULL	DATE + TIME	This is used for Automatic and Time Triggered Transitions. If the IN_PROGRESS status expires then the WorkItem will AWAIT_RETRY execution until the maximum retry is reached.
RETRY_COUNT	NOT NULL	NUMBER	Default is 0
PRIORITY_WEIGHTING	NOT NULL	NUMBER	Default is 1. The priority rating for this task. The higher the number the higher the priority.

			Time weighting is used here to work out the priority. Formula $\text{priority} = (\text{Current DATE} - \text{CREATED\_DATE}) * \text{PRIORITY\_WEIGHTING}$
EXPECTED_COMPLETION_DATE	NOT NULL	DATE + TIME	The estimated completion date. This is taken from the <estimatedCompletionTime/duration tag> in the example config.
LOCK_VERSION	NOT NULL	NUMBER	Default is 0
CREATED_DATE	NOT NULL	DATE	The time stamp when this case was created.
UPDATED_DATE	NOT_NULL	DATE	The time stamp of when this column was updated

#### 6.5.5 **TIME\_TRIGGER\_SCHEDULER**

This is only used if the OPERON\_TASK.TRIGGER\_TIME is not null. If the TASK has associated TIME\_TRIGGER\_SCHEDULERS then this implies we are using implicit timers otherwise its explicit timers.

1. Explicit - the Scheduler will execute straight away once expired.
2. Implicit - the Scheduler will use one of the registered schedulers and execute at the next round of execution round.

Column	Constraint	Type	Description
TASK_ID	FK	NUMBER	A Foreign key to the associated Case
SCHEDULER_REF	NOT_NULL	VARCHAR(50)	Points to a registered scheduler within the XML config file.
CRON_EXP	NOT_NULL	VARCHAR(80)	The Crontab expression. This is going to be used by the scheduler as the reference id to execute the WorkItem. E.g. If the registered scheduler with CronExp "0 0/2 * * * ?" then every time it executes it will search for all CRON_EXP that matches it and executes the WorkItem. This means only one scheduler for that particular time is created globally.
CREATED_DATE	NOT NULL	DATE	The time stamp when this case was created.
UPDATED_DATE	NOT_NULL	DATE	The time stamp of when this column was updated

#### 6.5.6 **OPERON\_EVENT\_AUDIT**

This table records all the events that update the status changes of a Case or a Task.

Column	Constraint	Type	Description
EVENT_AUDIT_ID	PK	NUMBER	The event audit primary key
CASE_ID	FK, Not Null	NUMBER	A Foreign key to the associated Case
TASK_ID	FK, Allow Null	VARCHAR(50)	A foreign key to the associated Task. If this is null then this implies that this is a Case event audit. If not null then this implies that this is a Task event

			audit.
EVENT	NOT NULL	VARCHAR(11)	<p>There are two event types of events Case events and WorkItem events: See Case Statuses and WorkItem Statuses diagram for events.</p> <p>Case Events</p> <ol style="list-style-type: none"> <li>1. NEW – creating a new Case</li> <li>2. CLOSE – to finish a Case</li> <li>3. M_CANCEL – manually cancels a Case</li> <li>4. EXPIRE – Expiring a Case because the time limit has reached.</li> <li>5. ERROR – Putting the Case into error due to an Activity (WorkItem in execution) error.</li> <li>6. M_SUSPEND – user manually suspends Case</li> <li>7. M_RESUME – user resumes a suspended Case</li> </ol> <p>WorkItem Events</p> <ol style="list-style-type: none"> <li>1. NEW – creating a new WorkItem</li> <li>2. FIRE – Token is fired by the WorkItem associated Trigger</li> <li>3. FINISH – to finish an activity (executing WorkItem)</li> <li>4. ERROR – puts the activity into ERROR because an error occurred while processing activity.</li> <li>5. TIMEOUT – puts the Activity into AWAIT_RETRY status because it has reached its execution time limit.</li> <li>6. RE_EXECUTE – automatically re-executes the awaiting retry activity.</li> <li>7. MAX_RETRY – puts the Activity into suspended state after the maximum retry is reached.</li> <li>8. M_RETRY – user manually puts the Activity to AWAIT_RETRY so that the Activity can be re-executed.</li> <li>9. OR_CANCEL – an implicit OR cancellation. See Transition Types for implicit OR.</li> <li>9. M_UNDO – user can manually undo a Manually Triggered Activity. All implicit OR redundant WorkItems associated with undo Activity will also be re-ENABLED.</li> </ol>
INITIAL_STATUS	NOT NULL	VARCHAR(11)	The initial status
FINAL_STATUS	NOT NULL	VARCHAR(11)	The final status
SUCCESS_IND	NOT NULL	NUMBER(1)	0=not success and 1 = success
RESOURCE_ID	NOT NULL	VARCHAR(50)	The resource_id that triggered this event.
ERROR_CODE	ALLOW NULL	VARCHAR(50)	The error code if this event resulted in error.
ERROR_DETAIL	ALLOW NULL	VARCHAR(500)	The error details if this event resulted in error

CREATED_DATE	NOT NULL	DATE	The time stamp when this case was created.
--------------	----------	------	--

### 6.5.7 **OPERON\_TOKEN**

Column	Constraint	Type	Description
TOKEN_ID	PK	NUMBER	The token id primary key
TOKEN_STATUS	NOT_NULL	VARCHAR(9)	Possible options are: 1) FREE – the produce token in the place is free 2) LOCKED – the Token in the Place is locked by an Activity. 3) CONSUMED – the Activity finishes and has consumed the Token. 4) CANCELLED – Token is no longer available.
LOCK_BY_TASK_ID	ALLOW NULL	NUMBER	The Task ID that has locked this Token. NULL if Token status is FREE or CANCELLED
LOCK_VERSION	NOT_NULL	NUMBER	DEFAULT 0
CREATED_DATE	NOT NULL	DATE	The time stamp when this case was created.
UPDATED_DATE	NOT NULL	DATE	The last updated date

### 6.5.8 **OPERON\_TOKEN\_PLACE\_REF**

This table is in support of subnets. With subnets a Place can be references to other subnet Places.

Therefore a Token in one Place indicates that the Token is also in the other reference Place and vice versa.

Column	Constraint	Type	Description
TOKEN_ID	FK, Not Null	NUMBER	The token FK key
CASE_ID	FK, Not Null	NUMBER	A Foreign key to the associated Case
PLACE_REF	Not Null	VARCHAR(50)	A reference a place in the Wfnet defined in the Operon configuration file.
PLACE_REF_TYPE	Not Null	VARCHAR(8)	There are 5 types of reference: SOURCE – Indicates that this is a starting place SINK – Indicates that this is an end place INTERMED – indicates that this is just a normal Place reference INREF – indicates that this is the id of an IN Place of a subnet OUTREF – indicates that this is the id of an OUT Place of a subnet

CREATED_DATE	NOT NULL	DATE	The time stamp when this case was created.
UPDATED_DATE	NOT NULL	DATE	The last updated date

#### 6.5.9 ***OPERON\_TOKEN\_ENABLED\_TASK***

This table contains the Tokens that cause the creation of the WorkItems (ENABLED Tasks).

This table is used to disable WorkItems (make it REDUNDANT) in the case of when the Token fired (removed from a Place).

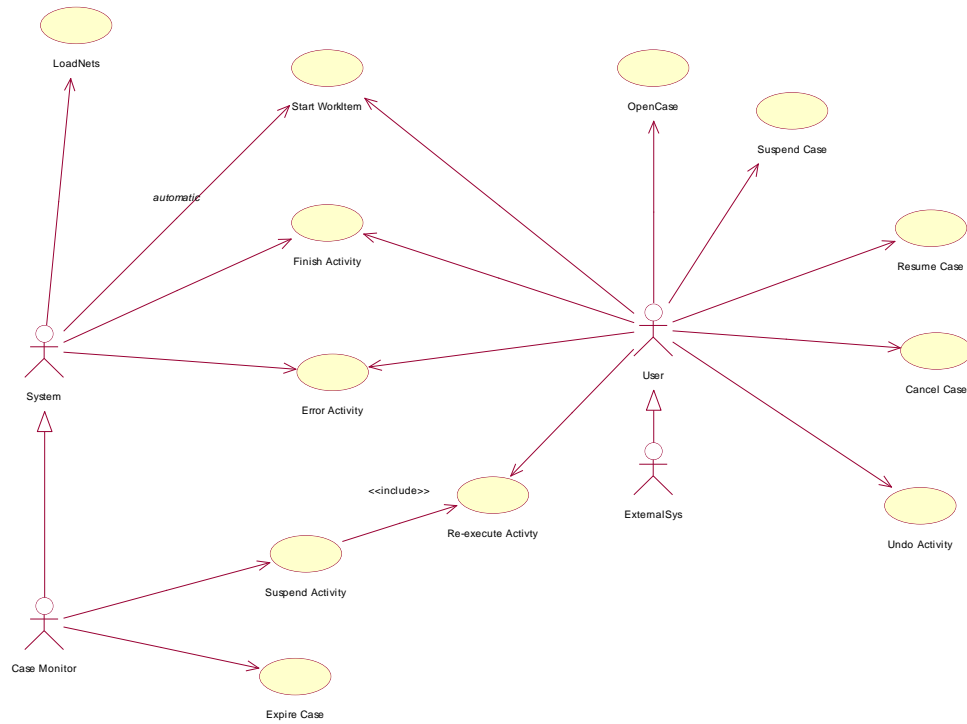
For example, in the case of an OR Transition, where there are two or more WorkItems but only one is allowed to be executed. The first WorkItem consumes the Token, the other one will be made REDUNDANT. The links between a token enabling many WorkItems will help us to determine which WorkItem is to be made redundant.

Column	Constraint	Type	Description
TOKEN_ID	FK, NOT NULL	NUMBER	The foreign key to the OPERON_TOKEN table
WORKITEM_ID	NOT_NULL	NUMBER	The foreign key to the OPERON_WORKITEM table
CREATED_DATE	NOT NULL	DATE	The time stamp when this case was created.

## 6.6

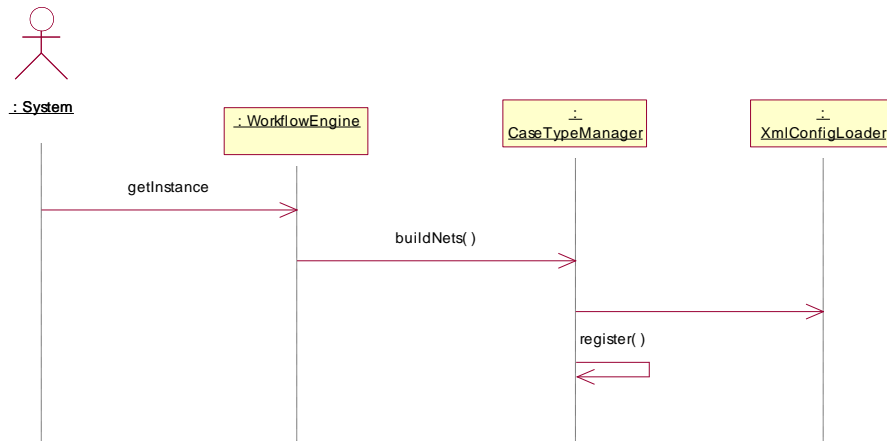
## Uses Cases And Sequence Diagrams

The diagram below shows the various possible use cases for Operon.

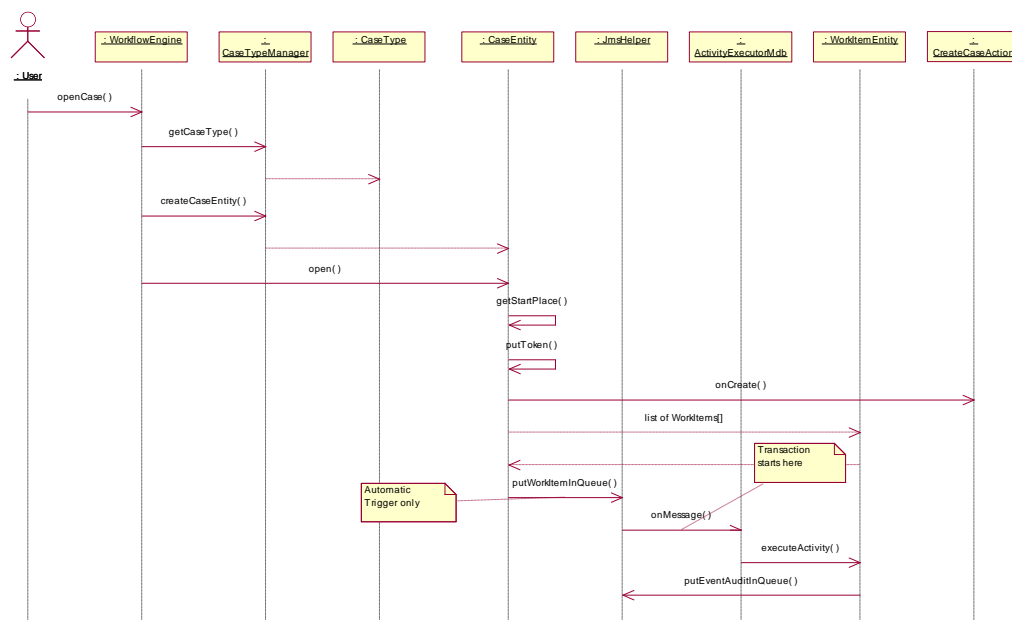


### 6.6.1 **Load Nets**

This use case shows how Operon builds the net from an XML file.

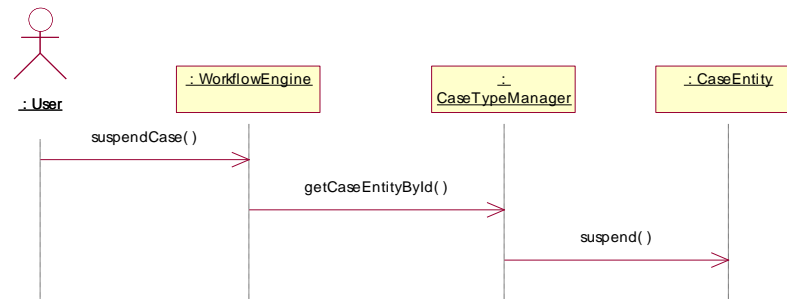


### 6.6.2 **Open Case**

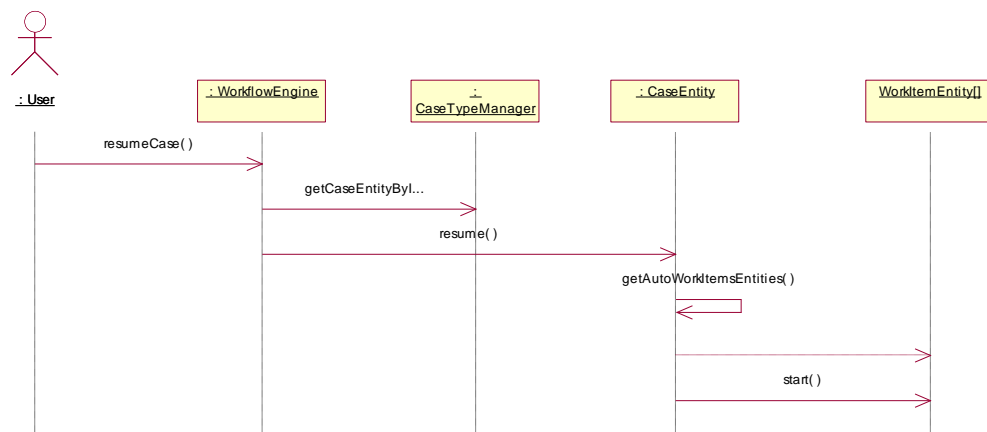


Note: Not shown but every status change results in an event audit being logged.

### 6.6.3 *Suspend Case*

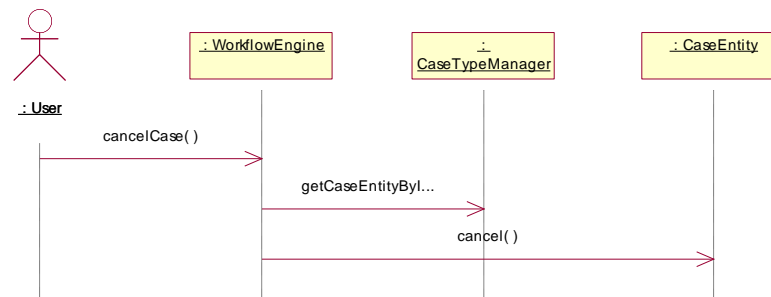


### 6.6.4 *Resume Case*

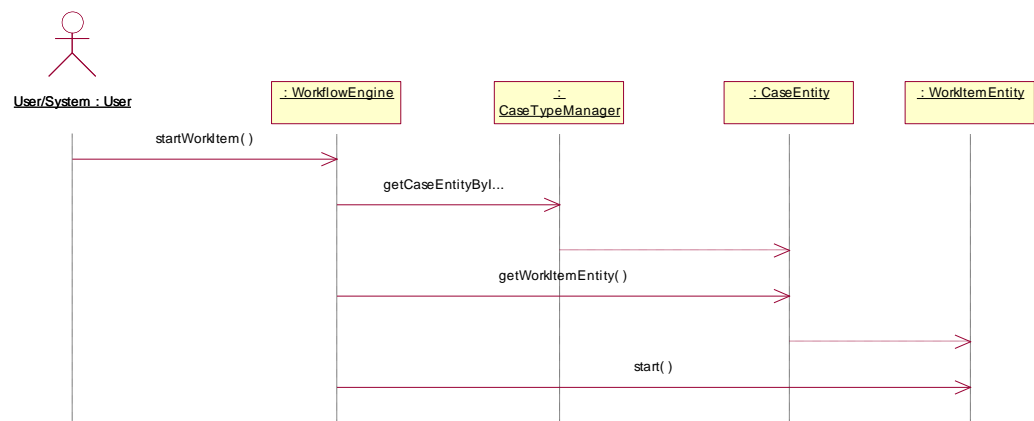




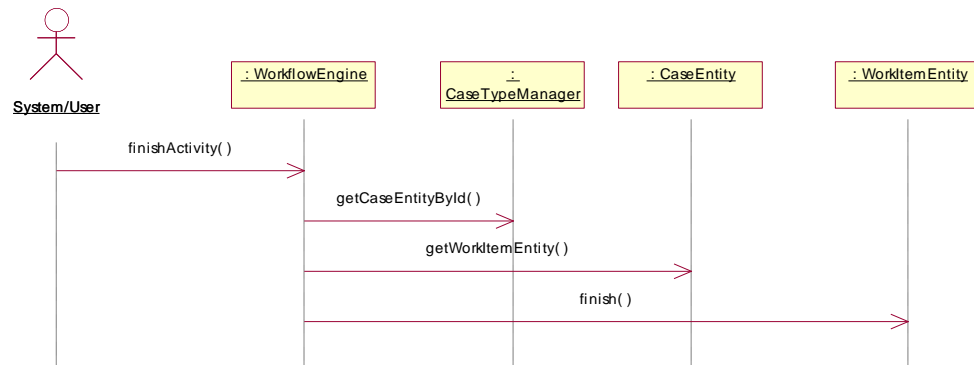
### 6.6.5 *Cancel Case*



### 6.6.6 *Start WorkItem*



### 6.6.7 *Finish Activity*



## 7 Developers Guide

### 7.1 Reusable Frameworks and Components

Name	Version	Description	Location
Spring Framework	1.2.6	Full stack J2EE Framework	<a href="http://www.springframework.org/">http://www.springframework.org/</a>
Apache Commons Components	N/A	Common reusable utility components	<a href="http://jakarta.apache.org/commons/">http://jakarta.apache.org/commons/</a>
iBATIS Data Mapper framework	2.1.7	Light weight Data Mapping framework for DAO	<a href="http://ibatis.apache.org/">http://ibatis.apache.org/</a>
Servteq Commons	1.0	Common reusable utility components	<a href="http://servteq.com/commons">http://servteq.com/commons</a>
Mysql	5.0	Database community edition	<a href="http://www.mysql.com/products/database/">http://www.mysql.com/products/database/</a>
Active MQ	3.22	JMS implementation	<a href="http://www.activemq.org/Download">http://www.activemq.org/Download</a>

### 7.2 Operon XML Configuration File