



Starting and Working with the  
*featherlite* Execution Rich Client Reference  
Implementation

Author: Michael Gatto  
Type: 5 minute tutorial  
Date: 2011-02-18  
Version: 0.1.0

## **Abstract**

The goal of 5 minute tutorial is to describe how to start and how to work with the reference implementation of the *featherlite* execution client. We also show what information is available to the user. By means of this tutorial, we also point out some of the features of the *featherlite* platform.

# Contents

<b>1 Overview and Aim of the Execution Plugin</b>	<b>1</b>
<b>2 Prerequisites</b>	<b>1</b>
<b>3 Starting and Stopping the Server and the Reference Client</b>	<b>1</b>
3.1 Starting the Server and the Client . . . . .	1
3.2 Shutting Down the Server and the Client . . . . .	2
<b>4 Working with the Reference Client</b>	<b>2</b>
4.1 Orders . . . . .	3
4.2 Resources . . . . .	4
4.3 Inspector View . . . . .	6
4.4 Gantt Chart . . . . .	6
4.5 Control View . . . . .	7
<b>5 Going further</b>	<b>7</b>

## 1 Overview and Aim of the Execution Plugin

This short tutorial is meant to be completed in 5 minutes. Here, we show how to start the server side and the client side of *featherlite*'s execution reference implementation. Then, we'll execute several orders of a predefined model, and show what data can be viewed from the client.

In Execution mode, *featherlite* allows to control sequences of actions, which start as a consequence of some order to be executed. These actions may involve the flow of some physical material through machines. The *featherlite* framework allows to control these actions from the business logic and all the way down to dispatching orders to the machines that perform the actions. For instance, *featherlite* can control the motion of a pallet through a network of carriers and conveyor belts: the user specifies an entry point and a destination for the pallet. The properly configured *featherlite* framework, in so-called execution mode, will then "talk" to the machines involved in the transport (which might even perform actions such as labeling the pallet with some pallet specific information, or move the goods on the pallet from one pallet to another one), get feed back from then, and go through the actions that brings the pallet to its defined destination, while keeping track of inventory and other aspects crucial to the business logic.

In this tutorial, we show a very simple such example, involving the transportation of pallets from one entry point to two exit points, by means of a network of three conveyors and one carrier.

## 2 Prerequisites

To be able to complete this tutorial, you need to have downloaded the *featherlite* Execution Reference Implementation from the website, [www.featherlite-framework.com](http://www.featherlite-framework.com). Keep in mind that as these are executable applications, you need to download the one specific to your platform (Windows, Linux or Apple, in the 32 or 64 bit version).

## 3 Starting and Stopping the Server and the Reference Client

We start with the basics: starting and stopping the execution server and client.

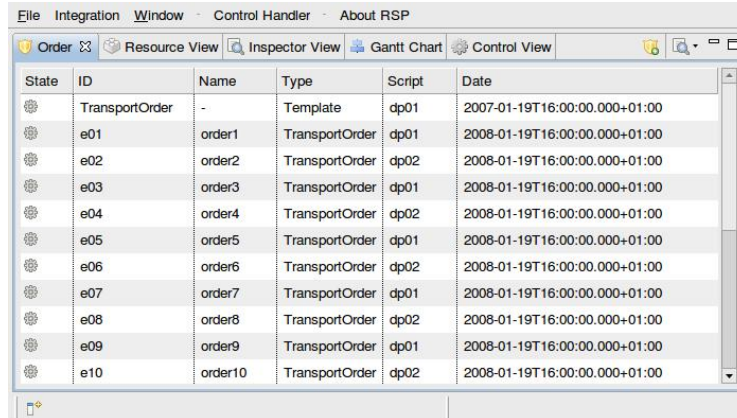
### 3.1 Starting the Server and the Client

First, start the server by simply double-clicking on the executable file called "Execution-Server"<sup>1</sup>. Depending on the platform, you may see a terminal opening and displaying the logging information of the server. If you cannot see any logging information, you can expect the server to be running within 5 to 15 seconds on current consumer hardware.

---

<sup>1</sup> If you have access to a terminal, you may also start the server from the command line.

To start the client, simply double-click on the executable file called “ExecutionClient”. Note that the server needs to be running at the client’s startup in order to ensure the correct communication between the client and the server. During startup, the client first displays a splash-screen; then, the user interface shown in Figure 1 appears.



State	ID	Name	Type	Script	Date
		TransportOrder	-	Template	2007-01-19T16:00:00.000+01:00
	e01	order1	TransportOrder	dp01	2008-01-19T16:00:00.000+01:00
	e02	order2	TransportOrder	dp02	2008-01-19T16:00:00.000+01:00
	e03	order3	TransportOrder	dp01	2008-01-19T16:00:00.000+01:00
	e04	order4	TransportOrder	dp02	2008-01-19T16:00:00.000+01:00
	e05	order5	TransportOrder	dp01	2008-01-19T16:00:00.000+01:00
	e06	order6	TransportOrder	dp02	2008-01-19T16:00:00.000+01:00
	e07	order7	TransportOrder	dp01	2008-01-19T16:00:00.000+01:00
	e08	order8	TransportOrder	dp02	2008-01-19T16:00:00.000+01:00
	e09	order9	TransportOrder	dp01	2008-01-19T16:00:00.000+01:00
	e10	order10	TransportOrder	dp02	2008-01-19T16:00:00.000+01:00

Figure 1: The reference execution client at startup.

If you could reach this point, you successfully loaded the execution reference client and the execution server.

### 3.2 Shutting Down the Server and the Client

To shut down the reference client, open the File menu, and select the “Shutdown server and client” menu entry. As is apparent from the menu description, this action shuts down both the server side and the client. In the dialog that appears you can confirm the shut down, or cancel the action.

## 4 Working with the Reference Client

Now that we have the application running, we proceed with analyzing an order, execute it, and see the repercussions of the execution steps on the resources affected by this order. We look at this in both a text-based and in a graphical form.

The model that loads at startup represents a transport system built by three conveyors and one Carrier, schematically shown in Figure 2. Pallets are to be transported from the entry point “Entry 1” to one of the two exit points “Exit 1” or “Exit 2”. To that aim, at the point “Entry 1” each pallet is loaded onto Belt Conveyor 01. Belt Conveyor 01 transports the pallet to the Carrier. At that point, the pallets are unloaded from the Belt Conveyor 01 onto the Carrier. Depending on the requested exit point, the Carrier transports the pallet either to “Belt Conveyor 02” or to “Belt Conveyor 03”. The pallet is then unloaded from the Carrier to the belt conveyor, and the belt conveyor transports the pallet to the exit point (“Exit 1” for “Belt Conveyor 02”, “Exit 2” for “Belt Conveyor 03”). Obviously, the Carrier can transport only one item at the time, and the belt conveyors can have one

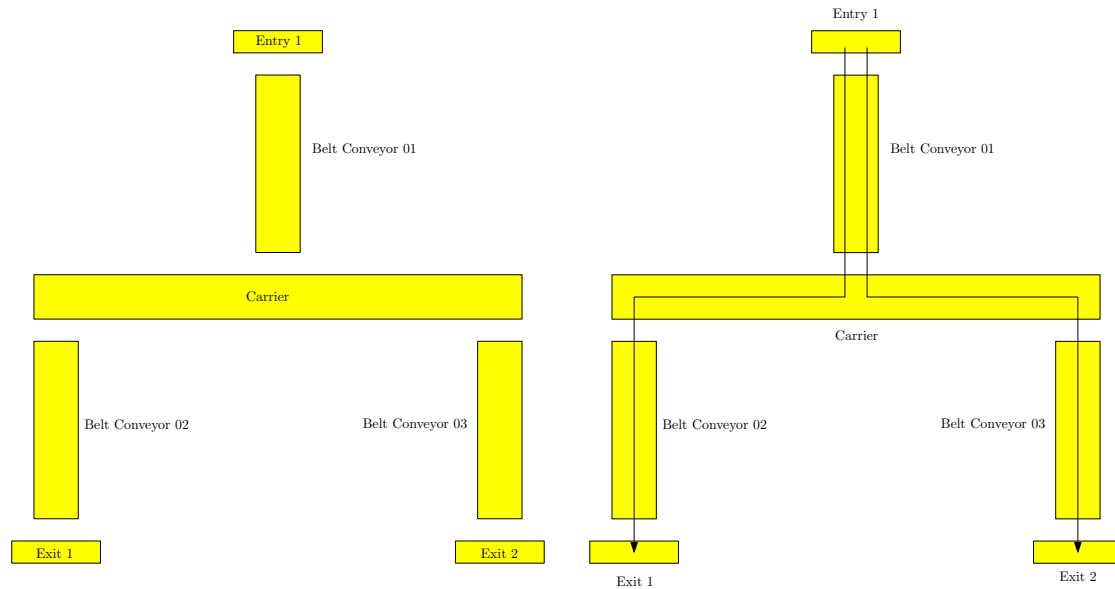


Figure 2: On the left, the structure of the conveyor network, and on the right hand side, the possible routes taken by the pallets through the network.

loading process at a time (since we cannot load or unload two pallets at the same time, but must do so sequentially).

## 4.1 Orders

Here, each order specifies a transport process, as for instance transporting one pallet of a specific length from “Entry 01” to the exit point “Exit 01”. These processes specify an execution process, meaning that while we execute them, we can trace the process over time. Hence, we are able to see in what “state” the order is: for instance, it might be being transported by some conveyor, or in the process of being loaded onto a conveyor.

In this view, orders are shown row-wise; for each order, we can see its execution state as an icon, its id, its name and its type. Moreover, we can see what script is used to execute the order. The script defines what actions are undertaken at execution, and thus what resources are affected by the order (Does the order occupy some conveyor?). Finally, each order specifies a date, which in execution mode serves only as an information and does not affect the execution in the standard configuration. In this model, the orders will be executed as soon as we tell them to execute.

We have a look at the orders by selecting order *e01*, opening the context-menu using right mouse button click and selecting the “Show Order Detail” menu entry. The two tabs show information stored on the order, as shown in Figure 3. In this order, we can for instance what quantity is to be produced with this order (1.0 pallets). The management data is empty.

We now proceed by executing this order. To this aim, select order *e01*, open the context menu by clicking on the order with the right mouse button, and select the “Execute”

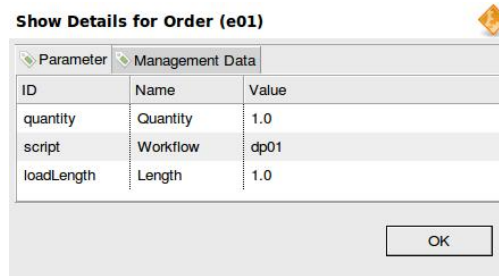


Figure 3: View of the details of an order. The orders' parameter specify a quantity (the number of pallets handled by this order), a loadlength (the length of each of the pallets), and a script (here, configured to "dp01", which selects the "Exit 01" as the pallet's destination).

menu entry. A clock should initially have appeared at the left of the order, in substitution of the red dot marker, and after some time (roughly 35 seconds) it is substituted by a green check mark, as shown in Figure 4.

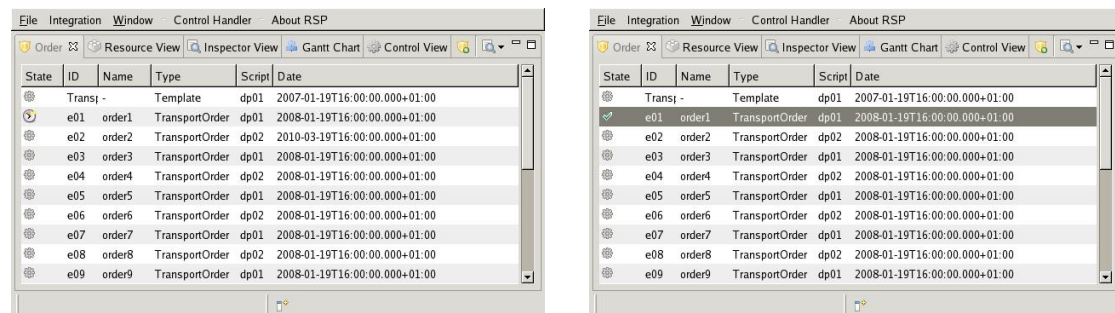


Figure 4: The order *e01*, which is executing in the left figure, and has completed its execution in the right figure.

Now, select the order *e01* and open the "Show Tasks" menu entry. Switch to the "All Tasks" tab, shown in Figure 5. Here, you can see the different steps of the execution process (technically called the *tasks* of the *workflows*). For instance, at first there is a task of type "Consume", which removes the pallets from the entry point. The second task is responsible for loading it onto the first conveyor belt (and is thus of type "Produce"). Next, we see that there is a task of type "Use", which represents the usage of the conveyor to transport the pallet from the source point to the Carrier. The rest of the process uses similar tasks, with the exception of the "Setup" task for the Carrier, which is responsible for bringing the Carrier from its current position to end of the "Conveyor Belt 01". If you look at the details while the order is being executed, you can see the different state of these tasks.

## 4.2 Resources

Let us switch to the "Resource View" tab of the client. Here, the different resources involved in this process are shown (the Entry and Exit points, the Carrier and the con-

**Show Workflow for Order (e01)**

ID	Name	Type	State	Resource	Start	End
consumeEntry01	consume from entry01	Consume	EXECUTED	Entry 1	2010-03-30T17:50:22.696+02:00	2010-03-30T17:50:22.876+02:00
produceConveyor1	produce on conveyor1	Produce	EXECUTED	Conveyor Belt 1	2010-03-30T17:50:22.825+02:00	2010-03-30T17:50:26.170+02:00
useConveyor1	use conveyor	Use	EXECUTED	Conveyor Belt 1	2010-03-30T17:50:26.172+02:00	2010-03-30T17:50:29.517+02:00
setupCarrier	setup carrier	Setup	EXECUTED	Carrier	2010-03-30T17:50:26.174+02:00	2010-03-30T17:50:31.185+02:00
consumeConveyor1	consume from conveyor	Consume	EXECUTED	Conveyor Belt 1	2010-03-30T17:50:31.188+02:00	2010-03-30T17:50:34.533+02:00
produceCarrier	produce on carrier	Produce	EXECUTED	Carrier	2010-03-30T17:50:31.189+02:00	2010-03-30T17:50:34.536+02:00
useCarrier	use carrier	Use	EXECUTED	Carrier	2010-03-30T17:50:34.537+02:00	2010-03-30T17:50:44.549+02:00
consumeCarrier	consume from carrier	Consume	EXECUTED	Carrier	2010-03-30T17:50:44.552+02:00	2010-03-30T17:50:47.897+02:00
produceConveyor2	produce on conveyor	Produce	EXECUTED	Conveyor Belt 2	2010-03-30T17:50:44.554+02:00	2010-03-30T17:50:47.900+02:00
useConveyor2	use conveyor	Use	EXECUTED	Conveyor Belt 2	2010-03-30T17:50:47.902+02:00	2010-03-30T17:50:51.247+02:00
consumeConveyor2	consume from conveyor	Consume	EXECUTED	Conveyor Belt 2	2010-03-30T17:50:51.249+02:00	2010-03-30T17:50:54.593+02:00
produceExit1	produce on exit 1	Produce	EXECUTED	Exit 1	2010-03-30T17:50:51.250+02:00	2010-03-30T17:50:51.264+02:00

Figure 5: The list of tasks which are associated to the workflow created by the order.

ID	Name	Type	Planning Policy	Placement Policy	Execution	Confirmation
carrier	Carrier	resource	-	CarrierPlacement	SimulatedExecution	InitializeAndRegisterConfirmation
conveyor0	Test Conveyor 00	resource	-	ConveyorPlacement	SimulatedExecution	InitializeAndRegisterConfirmation
conveyor1	Conveyor Belt 1	resource	-	ConveyorPlacement	SimulatedExecution	InitializeAndRegisterConfirmation
conveyor2	Conveyor Belt 2	resource	-	ConveyorPlacement	SimulatedExecution	InitializeAndRegisterConfirmation
conveyor3	Conveyor Belt 3	resource	-	ConveyorPlacement	SimulatedExecution	InitializeAndRegisterConfirmation
entry01	Entry 1	resource	-	SimplePlacement	SimulatedExecution	InitializeAndRegisterConfirmation
entry02	Entry 2	resource	-	SimplePlacement	SimulatedExecution	InitializeAndRegisterConfirmation
exit1	Exit 1	resource	-	SimplePlacement	SimulatedExecution	InitializeAndRegisterConfirmation
exit2	Exit 2	resource	NoPlanning	SimplePlacement	SimulatedExecution	InitializeAndRegisterConfirmation

Figure 6: The "Resource View" tab.

veyors). Beyond the IDs and names of the resources, this view (shown in Figure 6) also directly shows what type of policies are used by these resources. We'll dig into the policies in a specific tutorial.

We briefly look into the Carrier resource by selecting it, opening its context menu, and selecting the "Show Details" menu entry. If you select the "Bounds" tab, you can see the parameters bounding the moving speed of the Carrier, the capacity and the loading speed of the Carrier (see Figure 7, at left). These parameters can be configured and modified at runtime.

We can also see the changes in state of the Carrier. To this aim, after having closed the previous view, open the Carrier's context menu and select the "Show State Variables" menu entry. In the view, select the row entry "Velocity" on the left (see Figure 7, at right). This entry tells us about the speed the Carrier has over time. The "Actual State" of this so-called *state variable* reflects the current value of this state (so, if the actual state is 0, the Carrier is not moving). On the right hand side of this view, you can see the changes in speed the Carrier had while it was moving to fulfill the orders. In particular, you can

see that sometimes it moves at speed 0.002 (which is the “speed if empty” value we could also see in the “Bounds” dialog), sometimes it’s still (speed 0.0), and sometimes it moves at speed 0.001 (the maximum speed when loaded configured in the Carrier’s bounds). State variables show the modifications in state chronologically, and a specific state is valid from the time it is set until a new, later entry in the state variable appears.

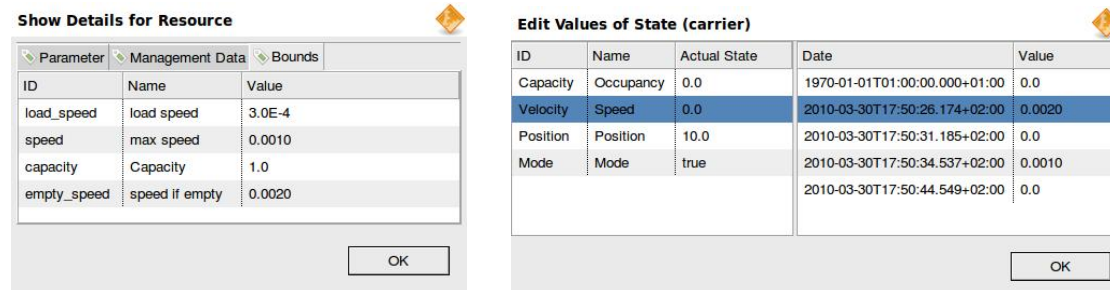


Figure 7: The Carrier resource. Left, we see the details of the configuration. On the right, we see the state variable for the speed, which effectively shows the speed of the Carrier over time.

### 4.3 Inspector View

The inspector view allows to browse through all objects instantiated at runtime. This view can be used to see all states of these objects, and is generally used by developers for debugging, or to see how to access the different fields of the object at hand. You can try and click around and see what data is displayed; however, an explanation of this view is beyond the scope of the 5 minutes.

### 4.4 Gantt Chart

In the Gantt Chart view, we see the development over time of the orders we plan. The tasks of the planned orders are shown as rectangles in the chart. To see how this view behaves, go back to the Order view, execute an order, and immediately return to the Gantt view. You should see rectangles (representing the tasks being executed) and triangles (representing loading and unloading activities) appearing over time (see Figure 8). Each such rectangle and triangle is located on the row of the resource it affects; the values of a resource’s state variables are shown directly below the resource. Thus, we can for instance see the speed of the Carrier as it changes over time. The values show changes over time similar as in the tabular view we saw before, in Section 4.2.

Now go back and select several orders and execute them at once. By switching back to the Gantt chart, you can see which tasks hinder each other from executing because they need exclusive access to the same resource. Also, you can see that the odd numbered orders will use “Belt Conveyor 02” and “Exit 01”, and the even numbered orders “Belt Conveyor 03” and “Exit 02”.

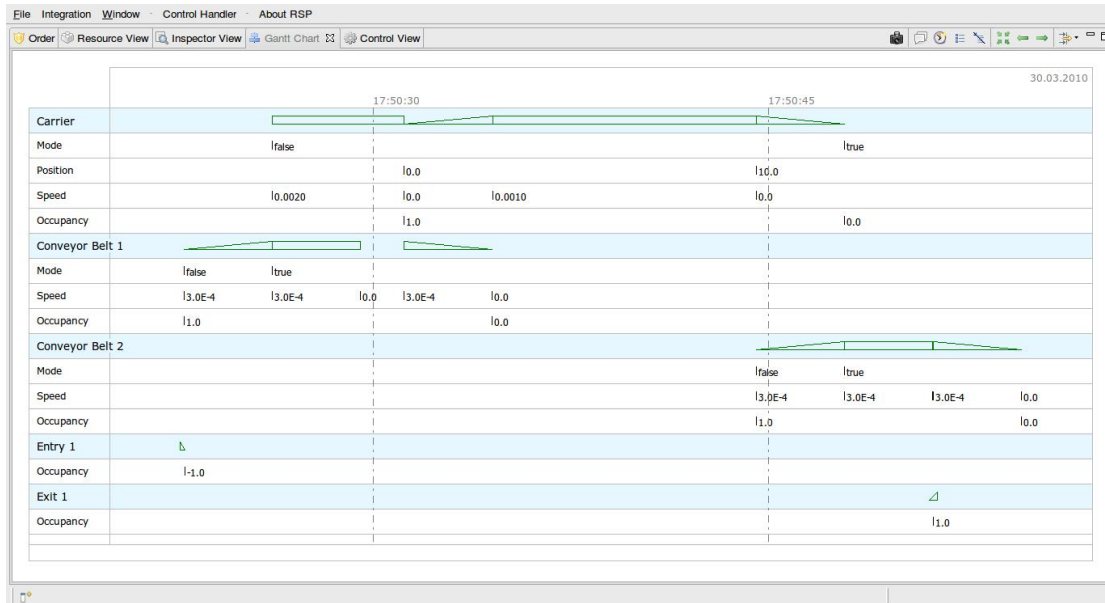


Figure 8: The Gantt Chart view, showing the tasks appearing over time. Note the solid vertical line moving rightwards (here already located in the right part of the chart), representing the point in time where we are now.

## 4.5 Control View

In the Control View, the *featherlite* reference execution client displays the state of the tasks of each order that has currently been planned to be executed. In Figure 9, for instance, we see that task “Use Carrier” is being executed (and on the right hand side, you see the task’s state in detail). Moreover, you can see that tasks of other orders have partially been executed and that some are now waiting for some resource to complete.

This view is particularly useful when we are dealing with processes coupled with real machines. In this view, we can see what information has been sent to the machines, and what the *featherlite* server is waiting for to proceed with the execution. Moreover, this view can be used to set the tasks’ states in case of an error in the facility which prevents the rest of the tasks from executing.

## 5 Going further

In this short tutorial, we explored some of the features of *featherlite* in execution mode, omitting how things work behind the scene. In our exploration, we used our reference client implementation, which is meant as a reference for your customized interfaces.

We looked at standard orders, which have an associated script defining the process that occurs during the order’s execution. *featherlite* allows to define many different types of orders, as of your needs. These orders can be individually configured with parameters and other data at runtime. Every order can be configured to use a different business

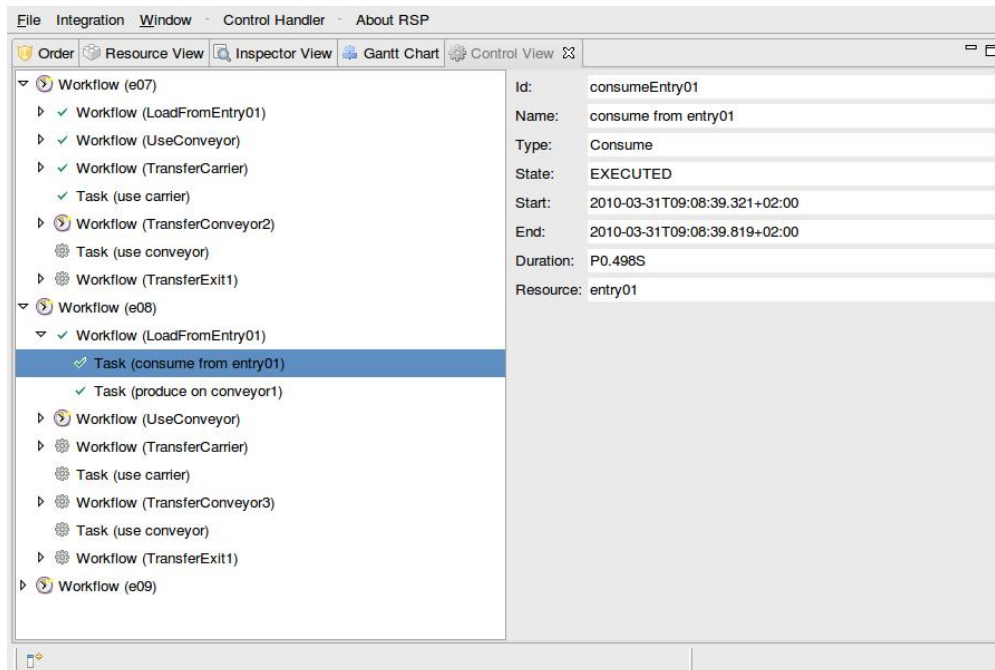


Figure 9: The Control View of the *featherlite* reference execution client. Here, we can see that some tasks have been executed, some are being executed and others wait for some resource to be freed.

logic: for example, in the model we used the script defined what exit the pallets needed to take. Moreover, the destination could also have been configured in the order, and the script instructed to take the destination from the order.

We also saw several different types of resources. The resources are again objects that can be configured at runtime with policies and other standard data. Here, we saw that conveyor belts and carriers, while both transporting a pallet, behave slightly differently. The *featherlite* framework offers several different pre-configured placement policies as for carriers and the conveyor belts, which can be configured to meet your specifications. The framework also allows to have these objects persisted into a database: the JEPlugin, which allows to persists the data into a BerkeleyDB database, is freely available for download from our website. The persistence plug-in using Hibernate (which can be configured to use the commonly used databases) can be purchased from the website.

This process simulated the act of a pallet passing through the system. Nothing really happened in practice, though, and everything was just a simulation. One of *featherlite*'s core functionalities is the possibility to communicate with machines through its Integration Framework. Thus, using *featherlite* you can really send a so-called telegram to a carrier, telling it to move from one position to the other, and wait for the carrier to move to the destination, get a confirmation from the carrier that it has reached the destination, and then instruct the carrier to take a pallet and move to another point. The execution structure and *featherlite* are built in such a way that communication with the outside world is not only possible, but is easy to achieve. Moreover, several communication protocols have already been implemented and are available as plug-ins. Of course, you

can build your own or ask us to build them for you.