

UNIVERSITY OF ROME 'LA SAPIENZA'
ENGINEERING FACULTY



“OPUS”

Service Integration

User Manual

Faculty of Engineering

Course in Computer Engineering

Authors: Balestra Concetta, Coccia Chiara, Colarullo Diego, Iachini Alberto, Qusa Hani

Promoter: Prof. Giuseppe De Giacomo

2009, 15th May

INDEX

Introduction	3
Chapter 1: How to manage services	
1.1 Available Service	5
1.2 Target Service	7
1.3 Cartesian product	8
Chapter 2:How application works	
2.1 Orchestator Table	11
2.2 Execution Orchestator	12
2.3 Orchestator Generator	14

2009, 15th May

Introduction

This manual helps users to understand how this application works. Orchestrator Application is developed with Java technology. To execute it need of Java Virtual Machine (JVM) version 1.6 or later. JVM, usually, is provided with operation system; if it is absent it can be downloaded freely from <http://java.sun.com/>. To run application there is a file .jar named orchestrator. Click on it and application will run.

Our project is based on the idea to transport composition via simulation algorithm into a java program. The composition concept requires other notions like Transition Systems, Service, Community, Orchestrator, bi-simulation and simulation. See documentation about these details.

After Application is run there will be this view

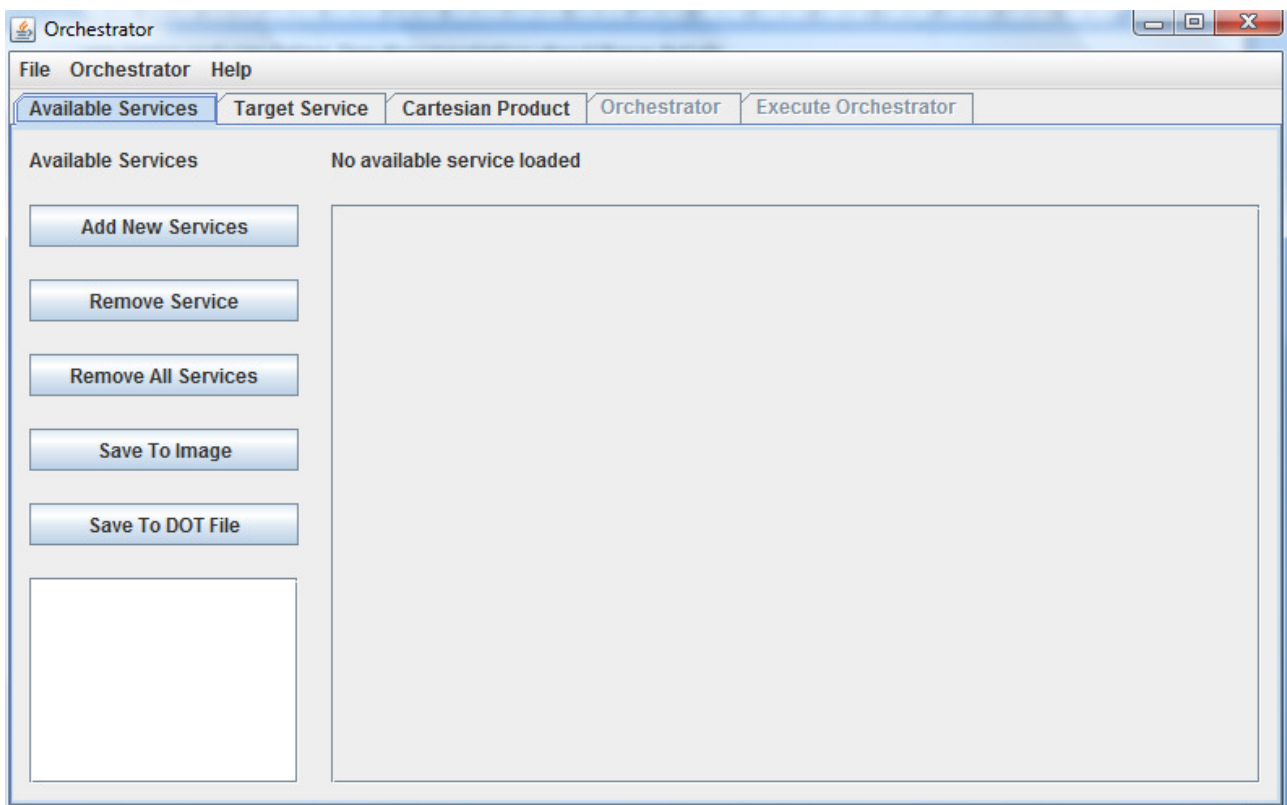


Figure 1: Application

We can see from the picture that there are four available tabs and two not available, in particular Orchestrator and Orchestrator Generator, because nor available and target services are loaded. After this operation these two tabs will become available (more details subsequently).

2009, 15th May

Chapter 1:

How to manage services

2009, 15th May

1.1 Available Service

This area is useful to manage available services. We have five options:

- ✓ *add new service*: to add an available service or more to community. Pressing on button new window will appear to search for file/s to load. It is possible to see all available service's name loaded in the rectangle below and selecting one of them the graph will appear on the right. Add new service option becomes not available after user decides to start orchestrator.

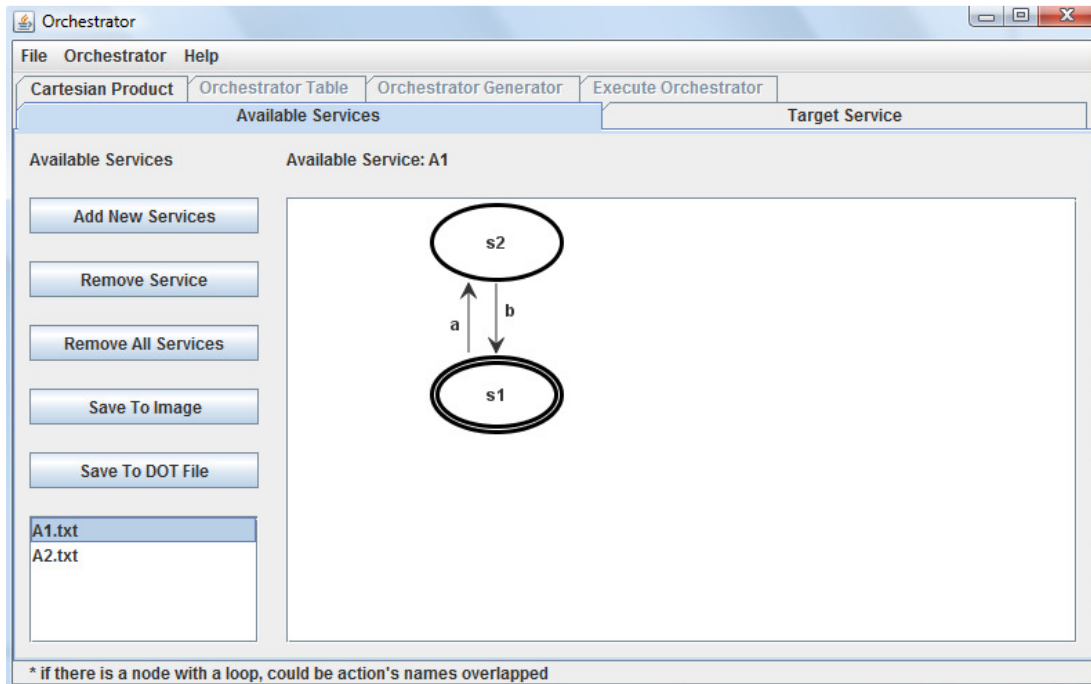


Figure 1.1 : Available services

- ✓ *remove service*: to delete a service, first select its name and after press delete button. It is possible to remove all services together pressing *remove all services*. These options become not available after user decides to start orchestrator.

The buttons explained above are the most important to use application and show orchestrator and simulation.

2009, 15th May

- ✓ Application provides other two buttons: one to save a graph in image, jpeg format, and another to save it in dot file. User can use one of these as he prefers. Pressing on button *Save to images or dot file* will appear a new window where user have to insert file's name and the extension .jpeg or .dot.

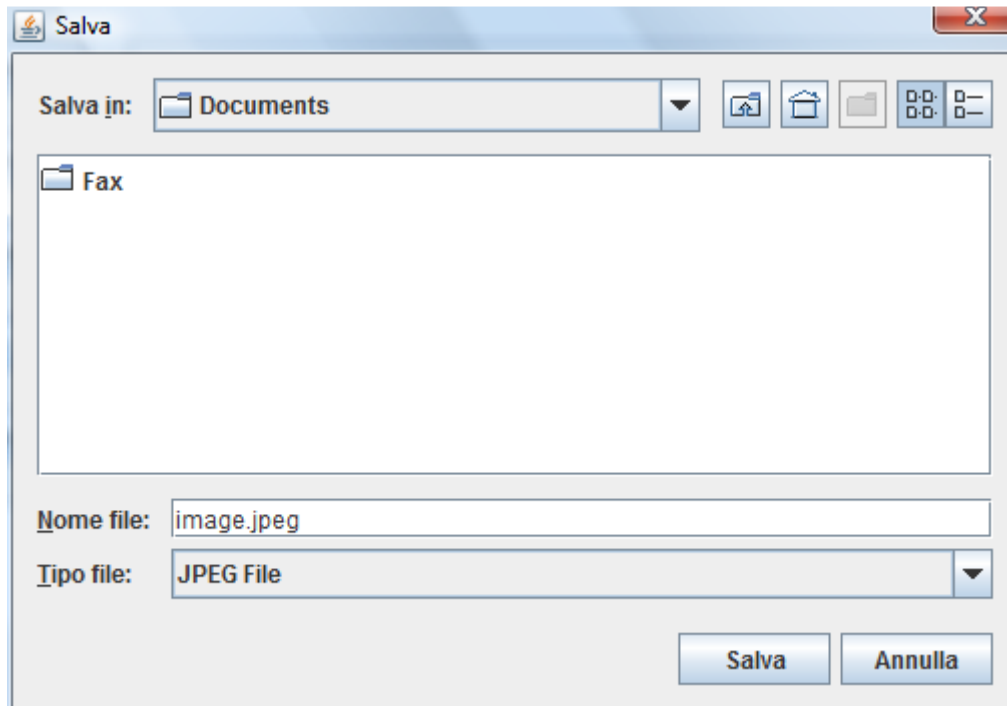


Figure 1.2: Save image

Above the white box, where is showed the graph, there is information about how many services are loaded in application, while selecting one of them there will be service's name.

1.2 Target Service

This view is similar to available service's window, in fact here it is possible to add a target service and show it like a graph on the right. There are the same options: “add a target service”, “remove a target service” and save service like image or like a dot file (in both case remember to add extension; .jpeg in first case, .dot in the second).

When a user tries to add a target, application checks if it is deterministic or not. In positive case it will be added and showed (*New target Service Loaded: nameFile*) otherwise a message will appear on the screen (in this case there is written *No target Service loaded*)

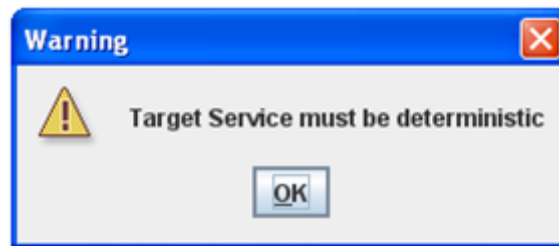


Figure 1.3: Warning target not deterministic

In both windows (available and target Service), when a user add a service/s and it isn't written according to formalism, application will notify to the user an error with some information:

- ✓ file in which there is the error;
- ✓ type error;
- ✓ error's line.

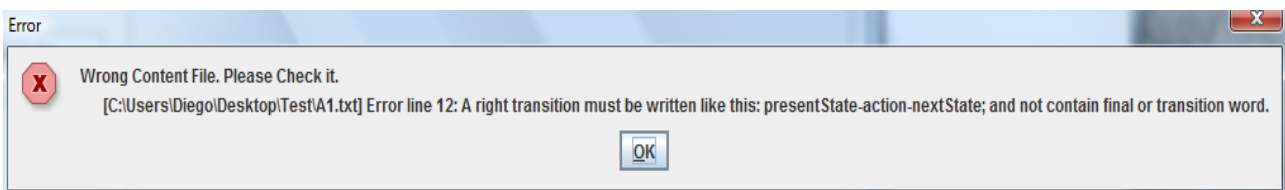


Figure 1.4: Error reading file

1.3 Cartesian product

In this view will be show a service representing Cartesian product among available services. Here there are four option, where two are always the same “*Save to image or dot file*” (remember to add a right extension) and other two are new.

- ✓ *Show Product*: if in community is present just one available service the product will show it, instead if there are at least two available services loaded, pressing this button, will appear a pop-up windows notifies to user the computational complexity of this product (more detail inside documentation) and asks him if he is sure to execute it. If user decides to execute it, application will show a graph representing Cartesian product.

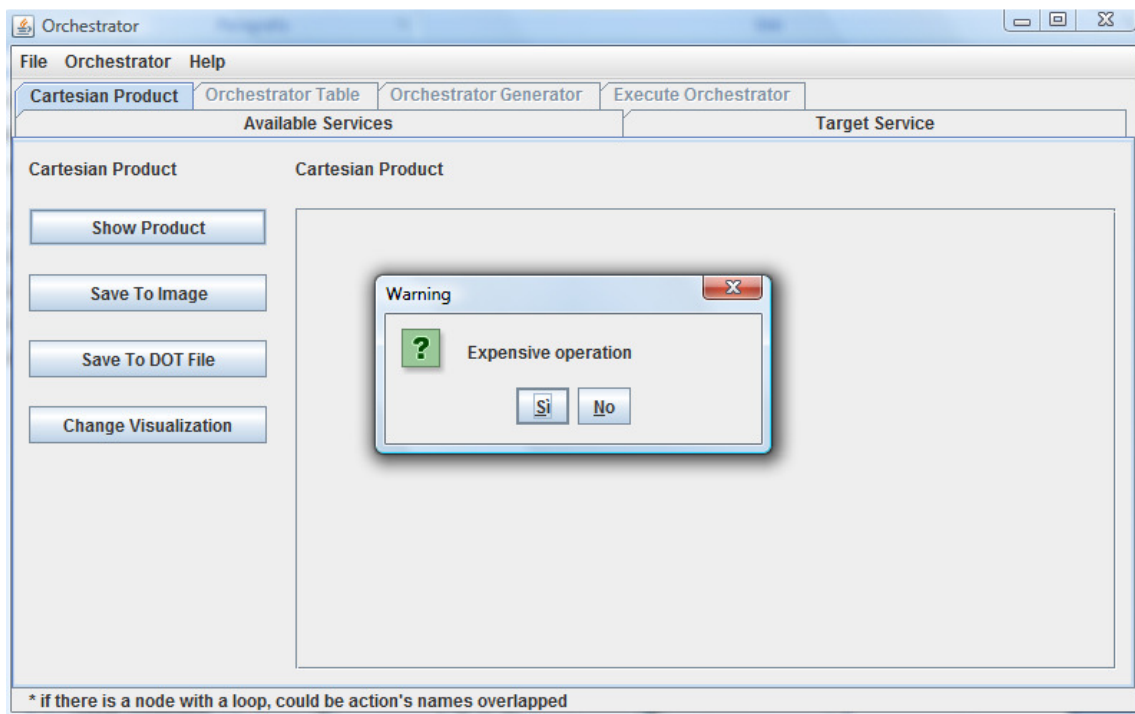
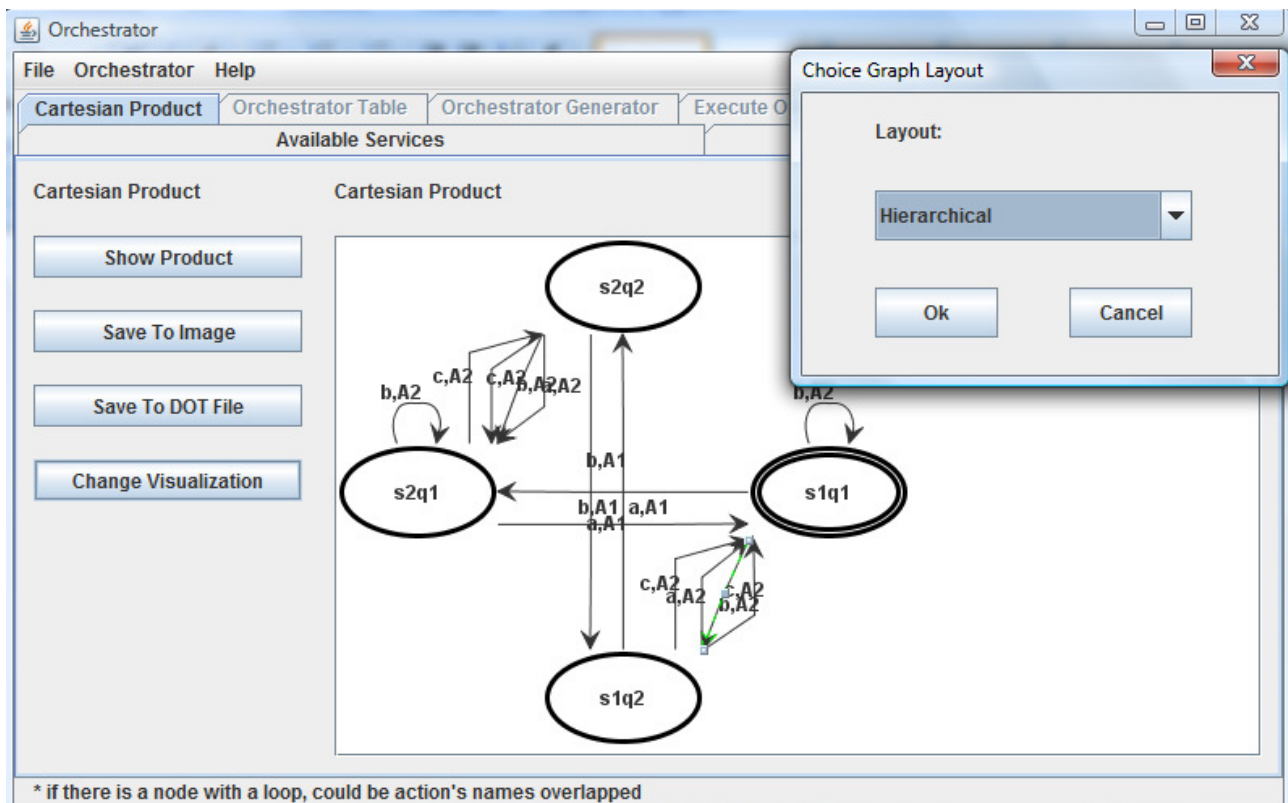


Figure 1.5: Cartesian Product

- ✓ *Change Visualization*: this button allows to the user to change the disposition of nodes in the area. It works only if there is a graph presents. Available options are:
 - Hierarchical: to have a representation top-down of nodes
 - Circle: to order all nodes in circle
 - Tree: to order all nodes to tree
 - Organic: to view a graph in organic way

2009, 15th May

*Figure 1.6: Cartesian Product Window*

2009, 15th May

Chapter 2:

How application works

2.1 Orchestrator Table

Orchestrator Table tab becomes available only when available and target services are loaded and the user decide to start Orchestrator (from menu Orchestrator). When User starts Orchestrator if *composition* doesn't exist, application will notify it through a pop-up window. In every case, it is always possible to show Orchestrator and Simulation.

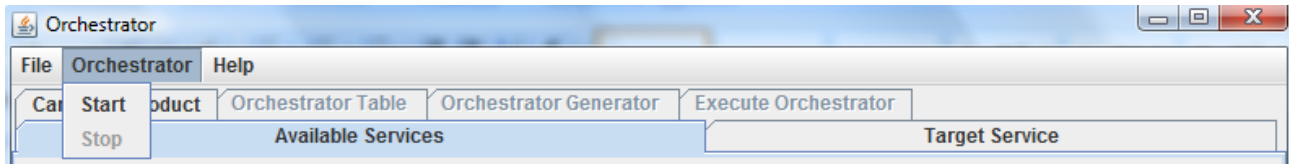


Figure 2.1: Start Orchestrator

In Orchestrator Table panel there are three options:

- ✓ *Show orchestrator*: pressing this button will appear a table on the right with the result of orchestrator. This table has four columns: one for the target state, one for the community state, one for action from target states and in the last there is/are available services able to perform this action.

Orchestrator Generator		Orchestrator		
<div>Show Orchestrator</div> <div>Show Simulation</div> <div>Save To Xml</div>		Target Service State	Available Services States	Action
		t0	x0,y0,z0	c
		t1	x0,y1,z1	b
		t1	x1,y0,z0	b
		t1	x0,y1,z0	b
		t0	x0,y0,z0	a
		t1	x1,y0,z1	b
		t0	x0,y0,z1	a
		t1	x0,y0,z0	b
		t0	x0,y0,z1	c
				Available Services
				As3,As2
				As2
				As1
				As2
				As1
				As1
				As3
				As2

Figure 2.2: Show Orchestrator

- ✓ *Show Simulation*: pressing this button will appear a table on the right with the result of simulation. This table has two columns: one for the simulated target state and another for community's state that simulates it.

Orchestrator Generator		Simulation	
<div>Show Orchestrator</div> <div>Show Simulation</div> <div>Save To Xml</div>		Simulated Target State	Is Simulated by (Community State)
		t0	x0y0z1
		t1	x0y1z0
		t1	x0y0z0
		t1	x1y0z1
		t0	x0y0z0
		t1	x1y0z0
		t1	x0y1z1

Figure 2.3: Show Simulation

2009, 15th May

- ✓ *Save to Xml*: this button is useful to save the result of simulation or orchestrator in xml format. Automatically, when user decides where to save the file, both xml and xls files will be saved. The last one is useful to add the style to xml file (the result is saved in a table like the one that appears in our application) so user doesn't need to know xml formalism. Before to save a xml file user has to choose what he wants to save between show simulation or show orchestrator.

2.2 Execution Orchestrator

In this view it is possible to see how Orchestrator works. In the left part of application there are different kinds of information. Let analyze them:

- ✓ at the top there is written the present state of target at each execution [Target present State name state];
- ✓ In the first small table there are two columns, one for available action from target's present state and another for next state reachable with this action;
- ✓ in the second table there are service's names able to perform the action selected from table above; switching the action in the first table these services will change.
- ✓ The table at the bottom has two columns, one for service's name and another to present the current state of each service.

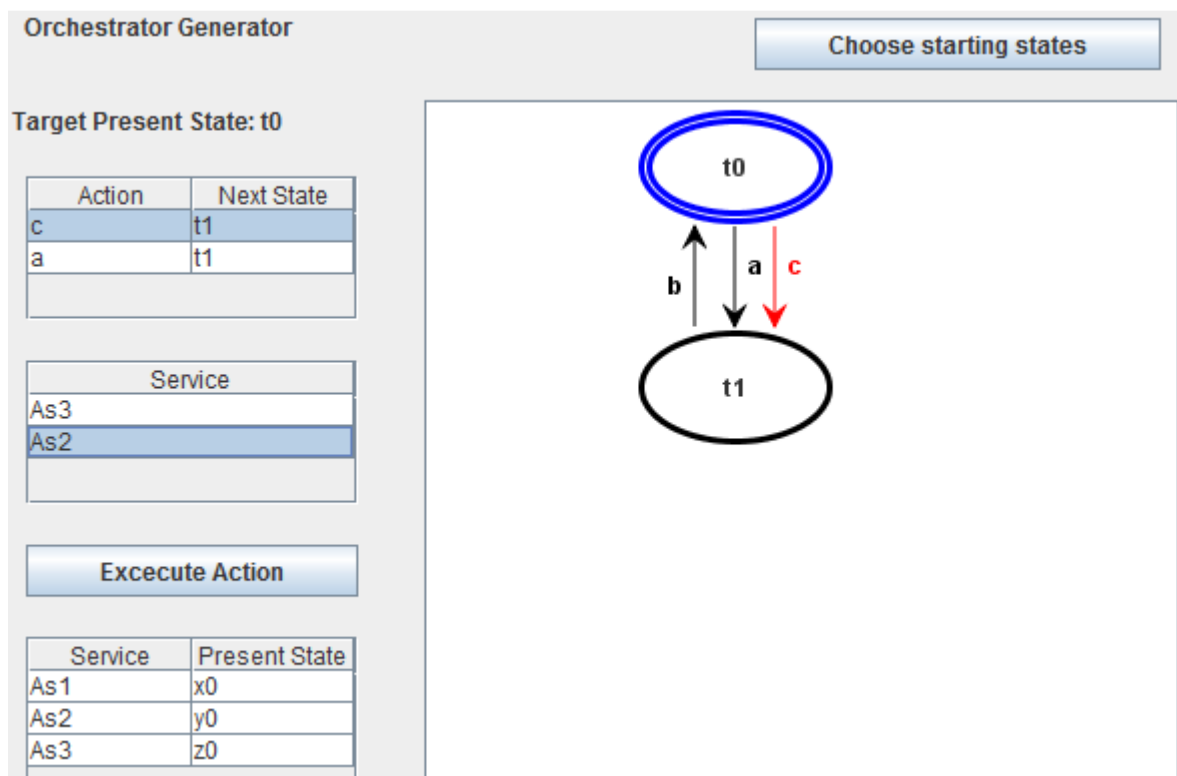


Figure 2.4: Execution orchestrator

2009, 15th May

- ✓ *Execute Action*: before pressing this button, an action from first table and a service from second table must be selected . After “*Execution*” target Present State will be t_1 (in this case). First table will be refresh according to the new target present state. In last table will be modified only the field (present state) relative to the service selected previously.

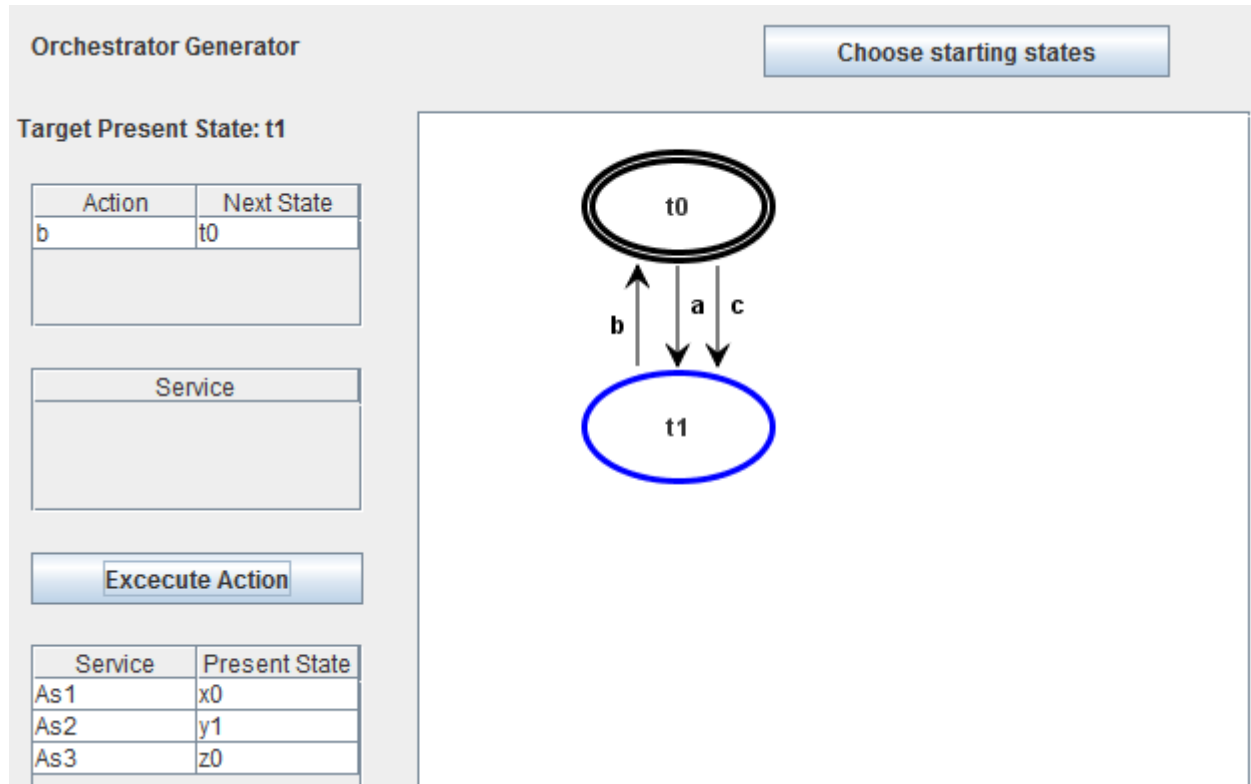


Figure 2.5: After Execute Action

- ✓ *Start Set States: (optional)* this button allows to change state (both target and available services loaded in application) from which we want to start the execution of orchestrator .

The 'Set Initial States' dialog box has a title bar with a close button. It contains three dropdown menus: 'Target Initial State: t0', 'Available Service Name:', and 'Available Initial States:'. At the bottom are 'Ok' and 'Cancel' buttons.

Figure 2.6: Set State

2.3 Orchestrator Generator

This tab becomes available when user decides to start orchestrator like *Orchestrator table* and *Execute orchestrator* tab. Here it is possible to see the graph of Orchestrator Generator.

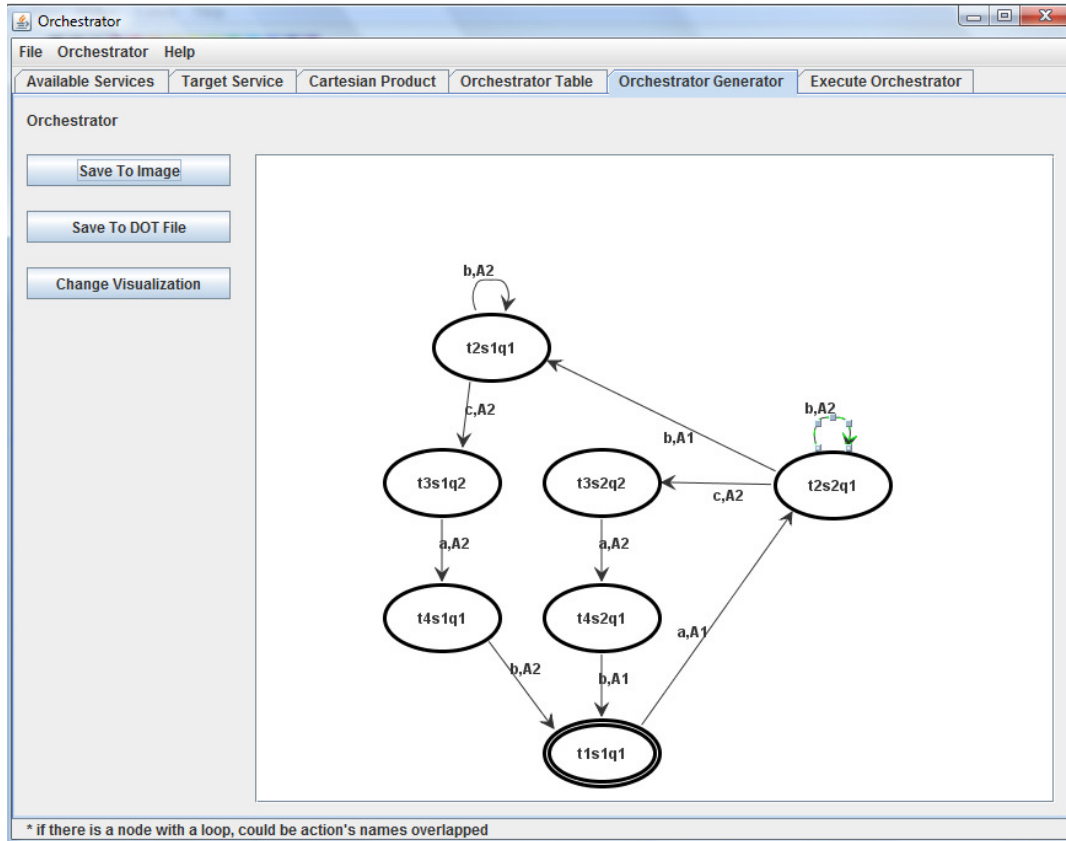


Figure 2.7: Orchestrator Generator View

User can decide to save graph or like image or dot file (remember to add extension before to save) or change visualization's kind of graph.

After all operations of Orchestrator, it is possible **to stop Orchestrator** (see fig 2.1) and these tabs (Orchestrator Table, Execute Orchestrator, Orchestrator Generator) became unavailable again but some buttons in *available and target service* tabs became available; in this way user can manage operations on available or target services like insert or remove it/them.