Instructions for using Angry Bots and JaCO

1. Start the JaCO Web Service locally, by opening the console, navigating to the folder of the project, and invoking:

```
java -jar jaco.jar
```

Instructions for using Angry Bots and JaCO

1. Start the JaCO Web Service locally, by opening the console, navigating to the folder of the project, and invoking:

```
java -jar jaco.jar
```

2. Start the Angry Bots patrolling domain executable by clicking twice on this icon:

Instructions for using Angry Bots and JaCO

1. Start the JaCO Web Service locally, by opening the console, navigating to the folder of the project, and invoking:

```
java -jar jaco.jar
```

2. Start the Angry Bots patrolling domain executable by clicking twice on this icon:

3. By pressing Start, you will request a new composition from the server, based on the current behaviors and target

Instructions for using Angry Bots and JaCO

1. Start the JaCO Web Service locally, by opening the console, navigating to the folder of the project, and invoking:

```
java -jar jaco.jar
```

2. Start the Angry Bots patrolling domain executable by clicking twice on this icon:

3. By pressing Start, you will request a new composition from the server, based on the current behaviors and target
4. By pressing Reset, you will return to the initial situation and use the last obtained composition again

The Angry Bots patrolling domain

- Developed specifically to accommodate the Behavior Composition model in a video game-like scenario
- Built upon the Unity game engine (Angry Bots technical demo)
- Specification of the domain:
  1. In the environment, we identified 20 points of interest (labeled with letters from A to T)
  2. Each NPC has a route leading it to some, but not all, the points of interest
  3. The NPCs' routes have overlaps: some points are covered by more than one NPC
  4. The target behavior we want to achieve is any desired patrolling routine (that may also include decision points)

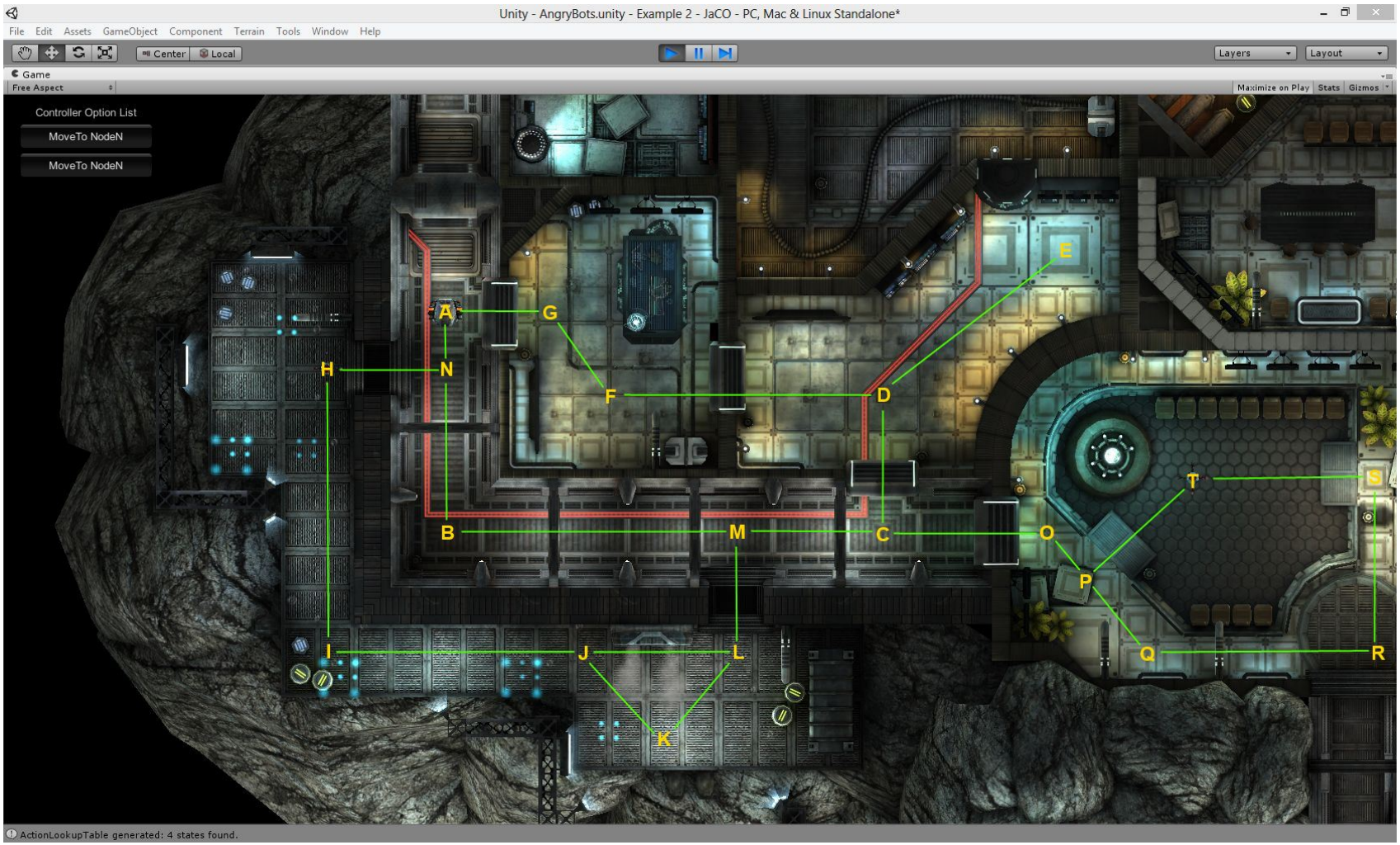
The Angry Bots patrolling domain

Relationship between the Behavior Composition framework and the Angry Bots patrolling domain:

- The **behaviors** are the finite state machines related to each of the non-player characters
- The **target behavior** is a desired collective behavior for the non-player characters (any patrolling routine)
- The **controller** is a computed control strategy that shows, for each possible situation, how each action can be realized and who can execute it

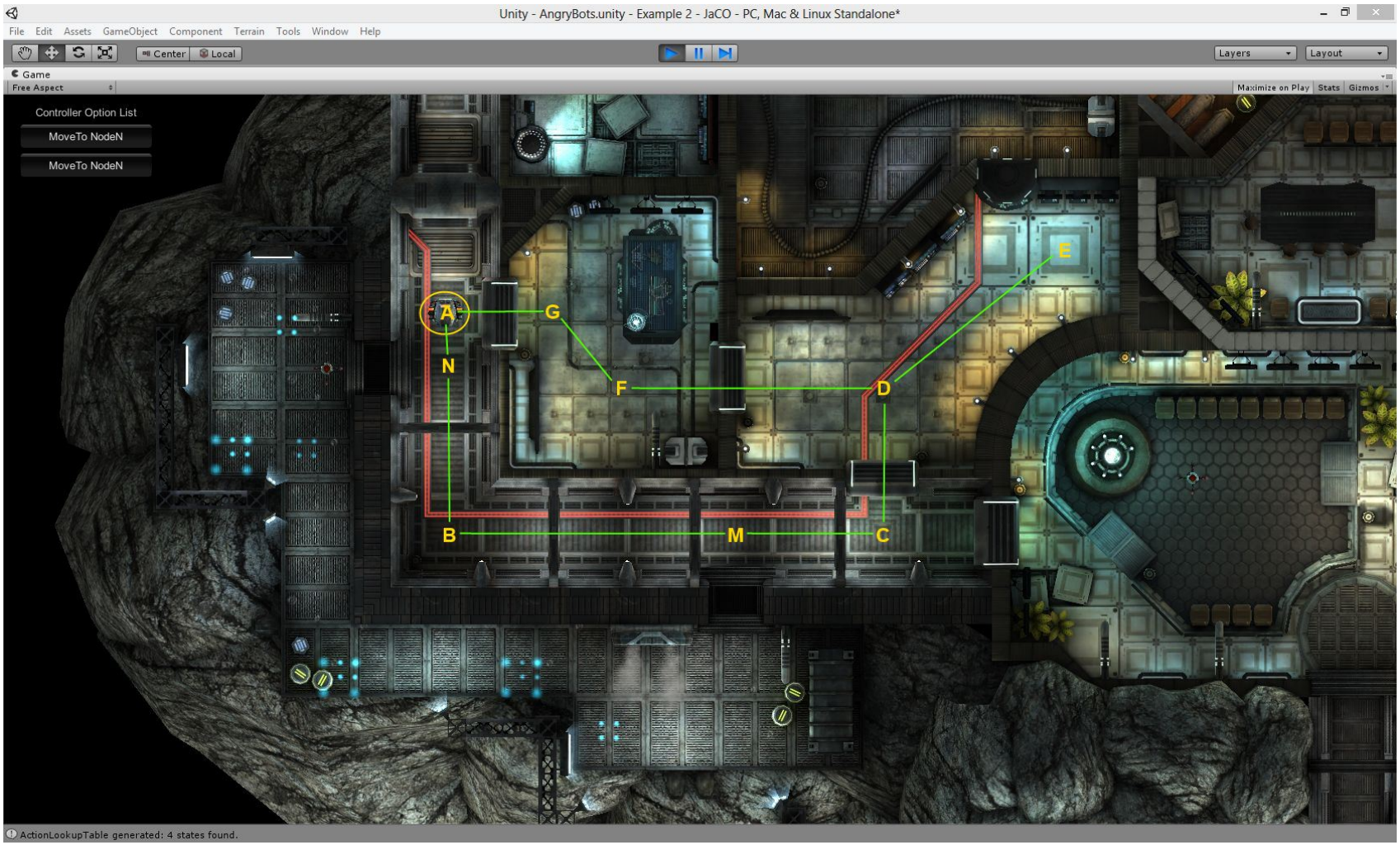
The environment and the available behaviors

Points of interest and connectivity network:



The environment and the available behaviors

MyEnemyMech route:



The environment and the available behaviors

MyEnemyMech route (expressed in Trivial Graph Format):

```
1 NodeA
2 NodeB
3 NodeC
4 NodeD
...
8 NodeM
9 NodeN
#
1 9 MoveTo
9 2 MoveTo
2 8 MoveTo
8 3 MoveTo
3 4 MoveTo
...
```

The environment and the available behaviors

MyEnemyMech route (expressed in Trivial Graph Format):

Declaration of nodes:

```
1 NodeA
2 NodeB
3 NodeC
4 NodeD
...
8 NodeM
9 NodeN
#
1 9 MoveTo
9 2 MoveTo
2 8 MoveTo
8 3 MoveTo
3 4 MoveTo
...
```

The environment and the available behaviors

MyEnemyMech route (expressed in Trivial Graph Format):

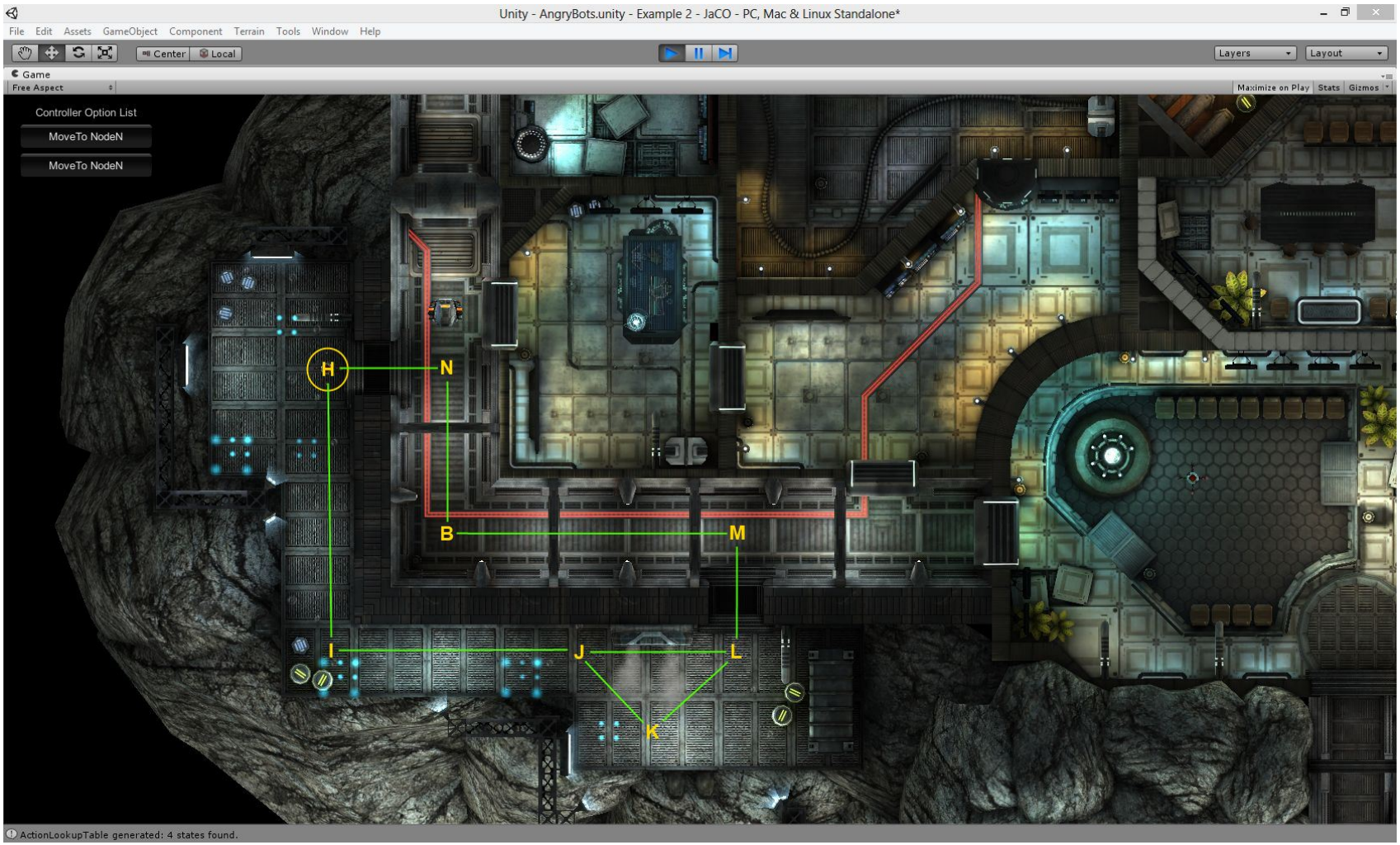
```
1 NodeA
2 NodeB
3 NodeC
4 NodeD
...
8 NodeM
9 NodeN
#
```

Declaration of edges

```
1 9 MoveTo
9 2 MoveTo
2 8 MoveTo
8 3 MoveTo
3 4 MoveTo
...
```

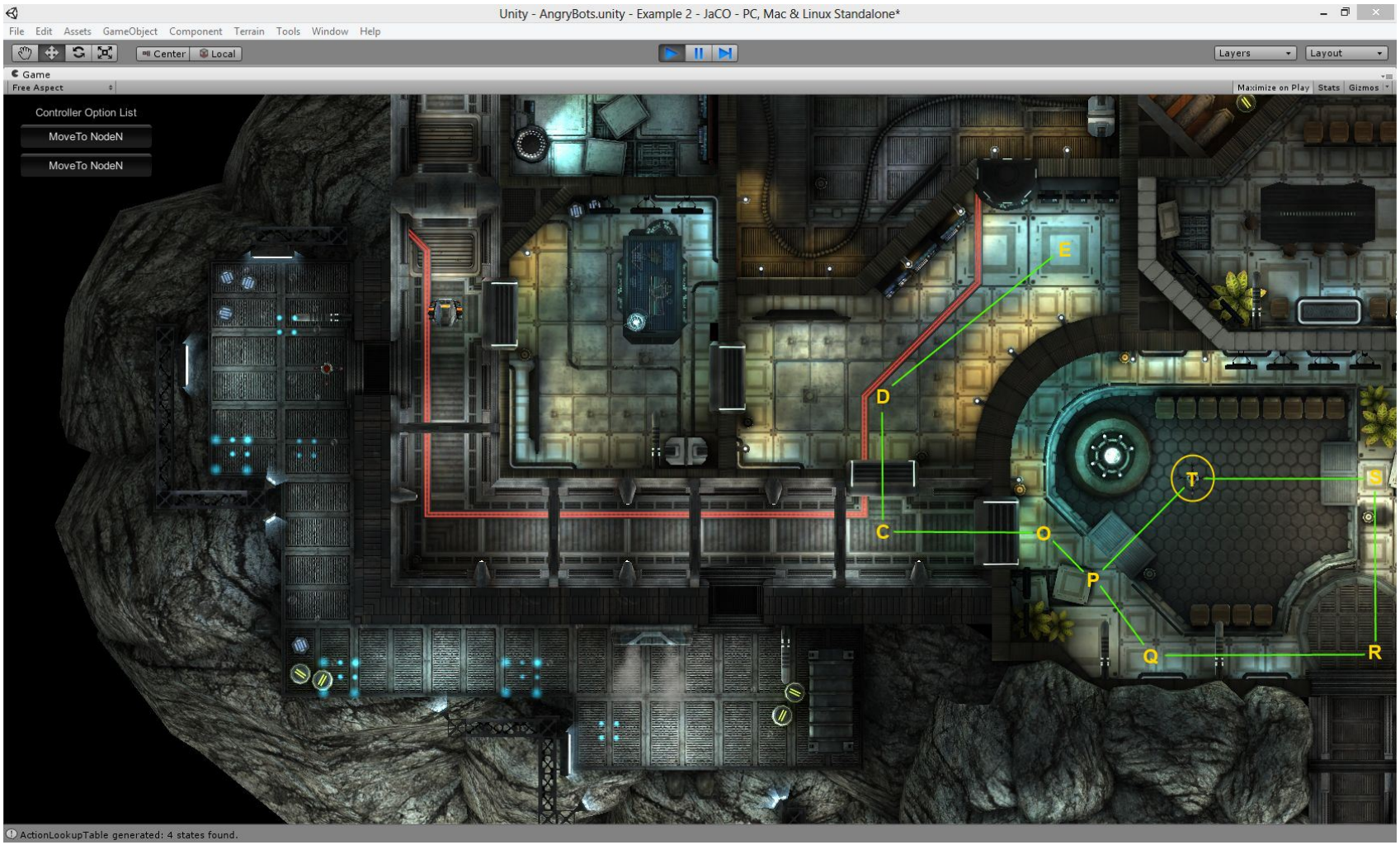
The environment and the available behaviors

MyEnemyMineBot route:



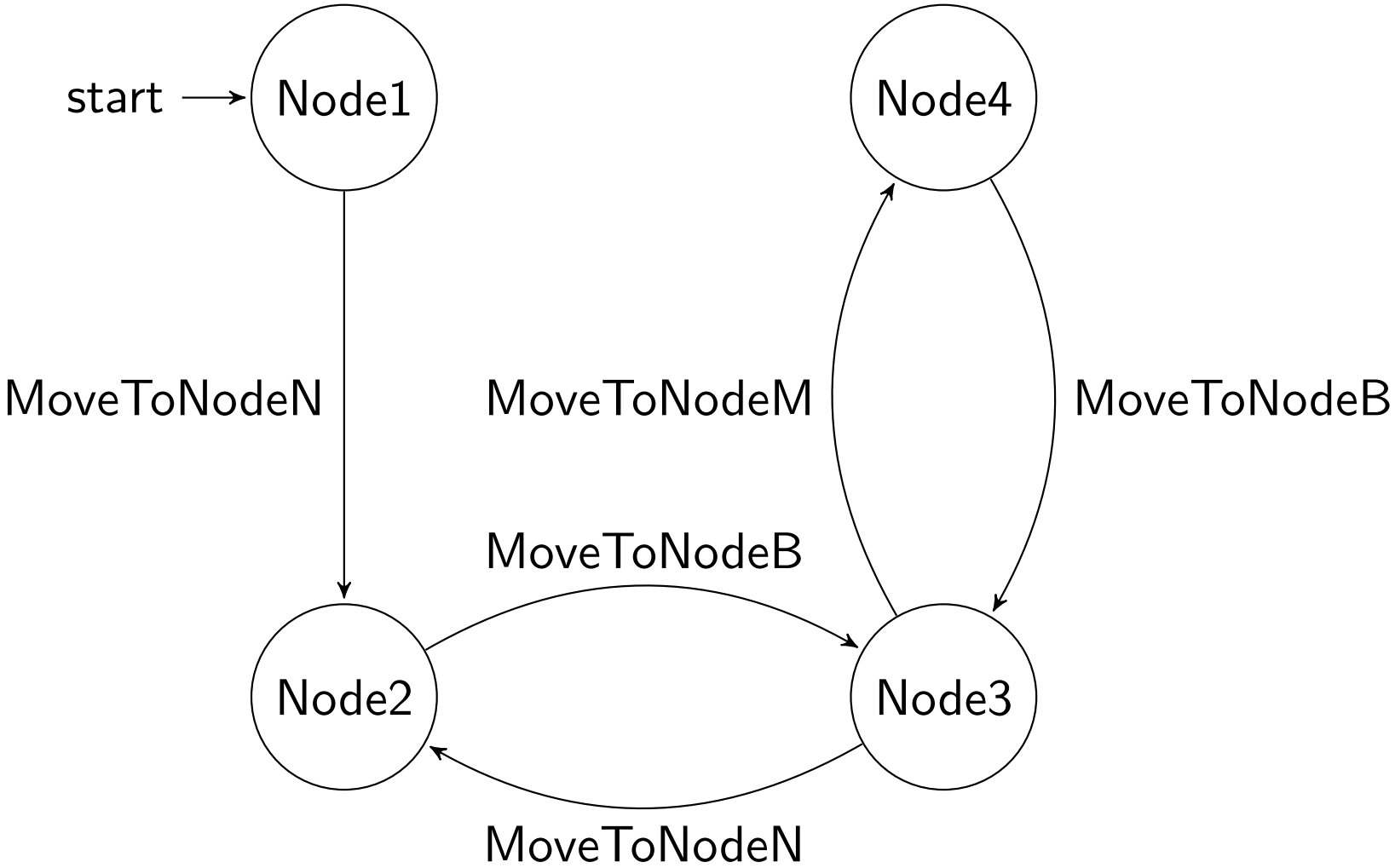
The environment and the available behaviors

MyEnemyMineBot1 route:



Example of target behavior

Expressed as a transition system:



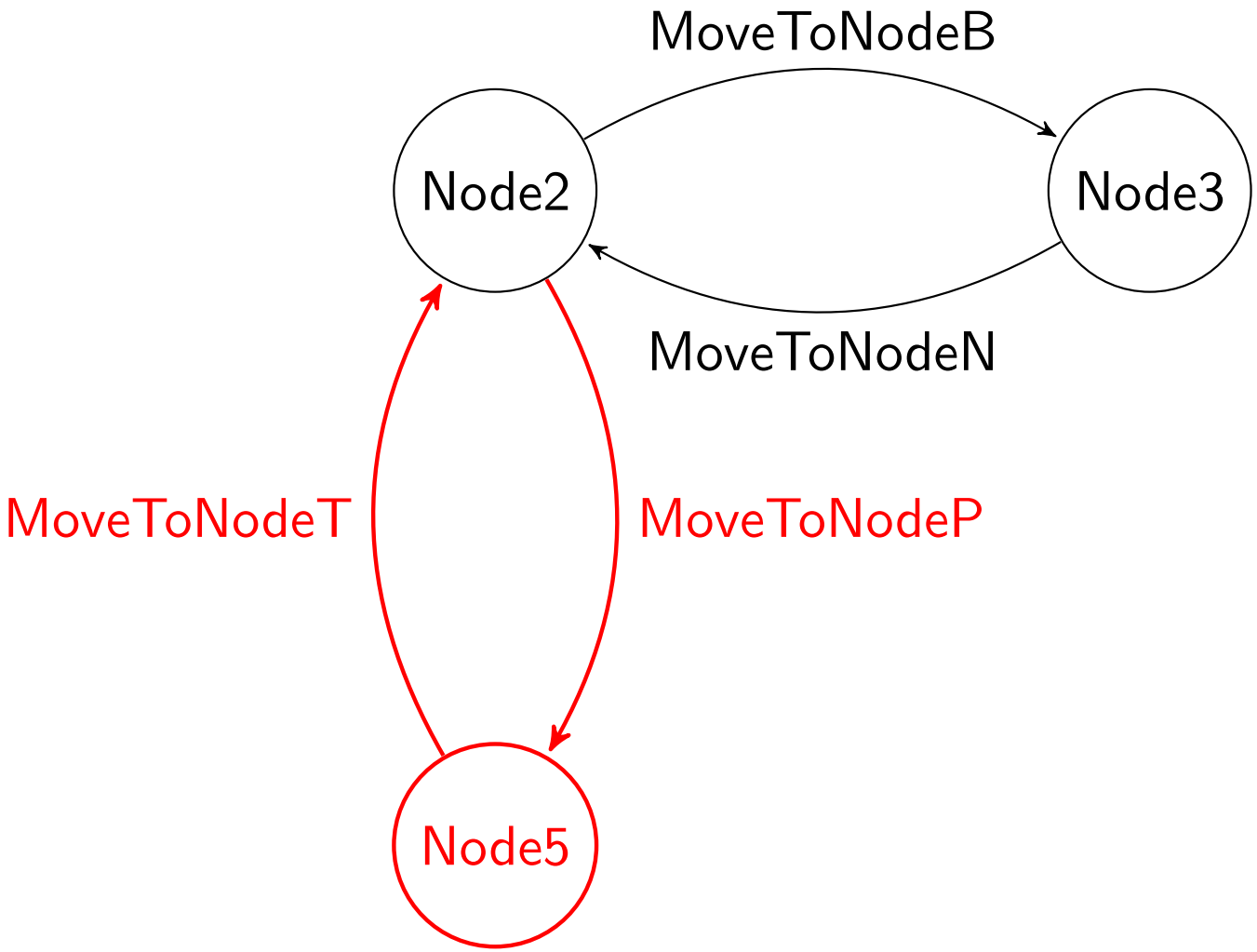
Example of target behavior

Expressed using the Trivial Graph Format:

```
1 Node1
2 Node2
3 Node3
4 Node4
#
1 2 MoveToNodeN
2 3 MoveToNodeB
3 2 MoveToNodeN
3 4 MoveToNodeM
4 3 MoveToNodeB
```

Example of target behavior

Let's add this slight modification to the target:



Example of target behavior

Let's add this slight modification to the target:

```
1 Node1
2 Node2
3 Node3
4 Node4
5 Node5
#
1 2 MoveToNodeN
2 3 MoveToNodeB
3 2 MoveToNodeN
3 4 MoveToNodeM
4 3 MoveToNodeB
2 5 MoveToNodeP
5 2 MoveToNodeT
```

The JaCO Web Service

- We designed a web service that provides behavior composition as-a-service, based on the REST principles: the whole interaction with the server is realized by sending and receiving HTTP messages

- The implementation of the service is called JaCO, which stands for **Ja**va-based **C**omposition-**O**riented Web Service

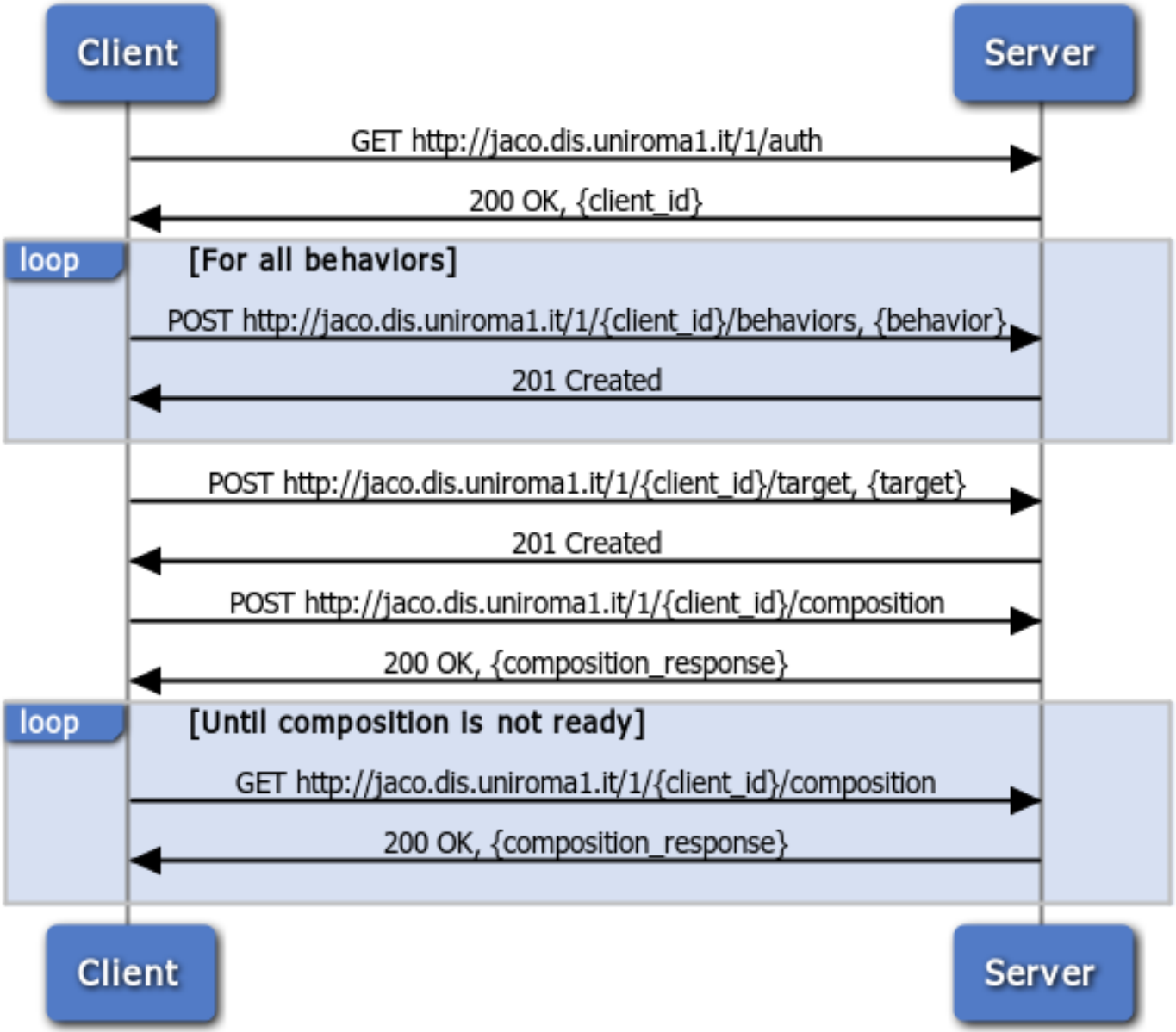
- The building blocks for the JaCO Web Service are:
  - **Apache Tomcat**, for deploying the API endpoints and managing the incoming HTTP connections;
  - **Jersey**, for developing a RESTful web application using the Java programming language;
  - **JTLV**, for calculating the composition by solving the safety game corresponding to the current problem instance
  - **Composition implementation**, provided by Alberto Iachini

The JaCO Web Service

Endpoints of the JaCO Application Programming Interface:

1. **/auth**: allows the user to retrieve the `client_id` that identifies him, and that he should communicate along the other requests
2. **/behaviors**: allows the user to send, retrieve, update or delete the finite state machines that define the behaviors
3. **/target**: allows the user to communicate the target behavior that he wants to be realized
4. **/composition**: allows the user to ask the server to compute the composition, and to retrieve it when it is ready

Usage scenario of the JaCO Web Service



A complete interaction with the JaCO Web Service

Step 1: Obtain the *client_id*

cURL command for interacting with JaCO:

```
curl http://jaco.dis.uniroma1.it/1/auth --request GET
```

A complete interaction with the JaCO Web Service

Step 2: Send the behaviors to the server MyEnemyMech route (expressed in XML):

```
<behavior>
  <name>MyEnemyMech</name>
  <finiteStateMachine>
    <state node="NodeA">
      <transition action="MoveToNodeN">
        <target>NodeN</target>
      </transition>
      <transition action="MoveToNodeG">
        <target>NodeG</target>
      </transition>
      <transition action="TakeSnapshotNodeA">
        <target>NodeA</target>
      </transition>
    </state>
    <!-- Other declarations of states -->
  </finiteStateMachine>
</behavior>
```

A complete interaction with the JaCO Web Service

Step 2: Send the behaviors to the server MyEnemyMech route (expressed in XML):

```
<behavior>
  <name>MyEnemyMech</name>
  <finiteStateMachine>
    <state node="NodeA">
      <transition action="MoveToNodeN">
        <target>NodeN</target>
      </transition>
      <transition action="MoveToNodeG">
        <target>NodeG</target>
      </transition>
...
```

cURL command for interacting with JaCO:

```
curl http://jaco.dis.uniroma1.it/1/client_id/behaviors
--request POST --header "Content-Type:text/xml"
--data @MyEnemyMech.xml
```

A complete interaction with the JaCO Web Service

Step 2: Send the behaviors to the server
Repeat the process for all the available behaviors...

cURL command for interacting with JaCO:

```
curl http://jaco.dis.uniroma1.it/1/client_id/behaviors
--request POST --header "Content-Type:text/xml"
--data @MyEnemyMineBot.xml
```

cURL command for interacting with JaCO:

```
curl http://jaco.dis.uniroma1.it/1/client_id/behaviors
--request POST --header "Content-Type:text/xml"
--data @MyEnemyMineBot1.xml
```

A complete interaction with the JaCO Web Service

Step 3: Send the target behavior to the server

```
<behavior>
    <name>TargetBehavior</name>
    <finiteStateMachine>
        <state node="Node1">
            <transition action="MoveToNodeN">
                <target>Node2</target>
            </transition>
        </state>
        <state node="Node2">
            <transition action="MoveToNodeS">
                <target>Node3</target>
            </transition>
        </state>
        <state node="Node3">
            <transition action="MoveToNodeI">
                <target>Node4</target>
            </transition>
        </state>
        <!-- Other declarations of states -->
    </finiteStateMachine>
</behavior>
```

A complete interaction with the JaCO Web Service

Step 3: Send the target behavior to the server

```
<behavior>
    <name>TargetBehavior</name>
    <finiteStateMachine>
        <state node="Node1">
            <transition action="MoveToNodeN">
                <target>Node2</target>
            </transition>
        </state>
        <state node="Node2">
            <transition action="MoveToNodeS">
                <target>Node3</target>
            </transition>
        </state>
        ...
```

cURL command for interacting with JaCO:

```
curl http://jaco.dis.uniroma1.it/1/client_id/target
--request POST --header "Content-Type:text/xml"
--data @Target.xml
```

A complete interaction with the JaCO Web Service

Step 4: Ask the server to compute the composition

cURL command for interacting with JaCO:

```
curl http://jaco.dis.uniroma1.it/1/client_id/composition
--request POST
```

Step 5: Poll the server to get the computed calculation, if ready

cURL command for interacting with JaCO:

```
curl http://jaco.dis.uniroma1.it/1/client_id/composition
--request GET
```