SAPIENZA
UNIVERSITÀ DI ROMA

Non-Player Character Behavior Composition in Unity Game Engine

Stefano Cianciulli

March 20th, 2013

Overview

- Motivation
- The Behavior Composition problem
- Adopted Technology
- The Angry Bots patrolling domain
- Behavior Composition using SM4LL
- The JaCO RESTful Web Service
- Future work

Motivation

- In modern video games, usually non-player characters are provided an "intelligence" using **reactive methods** for behaviors (e.g., Finite State Machines or Behavior Trees). These techniques decompose the behavior of NPCs in:
  - **states**: possible states of "mood" or "attitude"
  - **transitions**: actions or conditions that cause the character to change its "internal" state
- The AI technique called **behavior composition** allows to combine behaviors (expressed as FSMs) to coordinate them according to a desired collective behavior

The motivation of this work is to investigate how behavior composition can be employed to coordinate non-player characters within a video game scenario

Motivation

- Over the last few years, the Service-Oriented Computing approach is increasingly being adopted
  - Companies started to share algorithms and data by exposing web services according to RESTful principles
- The REST style is the one upon which the World Wide Web is built:
  - The key abstraction of information is a *resource*
  - Each resource has a *unique* identifier
  - Resources are interconnected by *hyperlinks*
  - Interactions are conducted by using the HTTP protocol

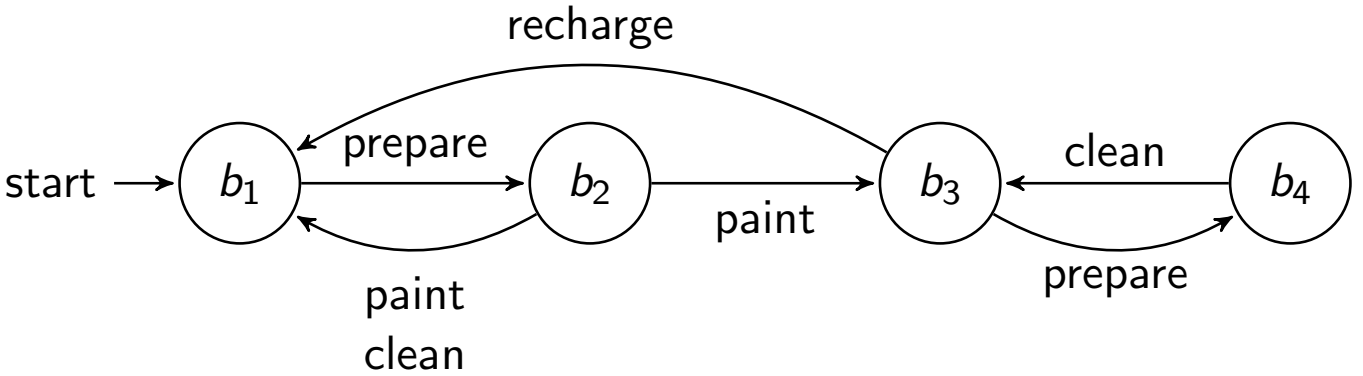
In this work, we follow this emerging trend by exposing the behavior composition method as a RESTful Web Service

The Behavior Composition problem

Inputs:

- A library of available **behaviors**

A behavior represents a program for an agent, or the logic of some device. Behaviors are modeled using *transition systems*:



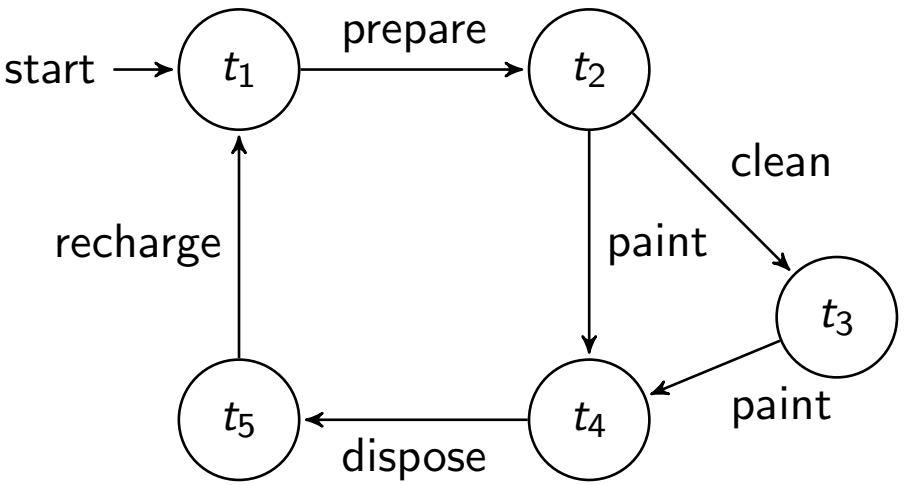
Behaviors are in general *non-deterministic*: the execution of action a in state s may produce *more than one* outcome.

The Behavior Composition problem

Inputs:

- A library of available **behaviors**
- A **target** behavior

The target behavior is a *deterministic* behavior over the environment.
It represents the *fully controllable* desired behavior to be obtained through the available behaviors.
The target behavior acts like a *virtual* component.

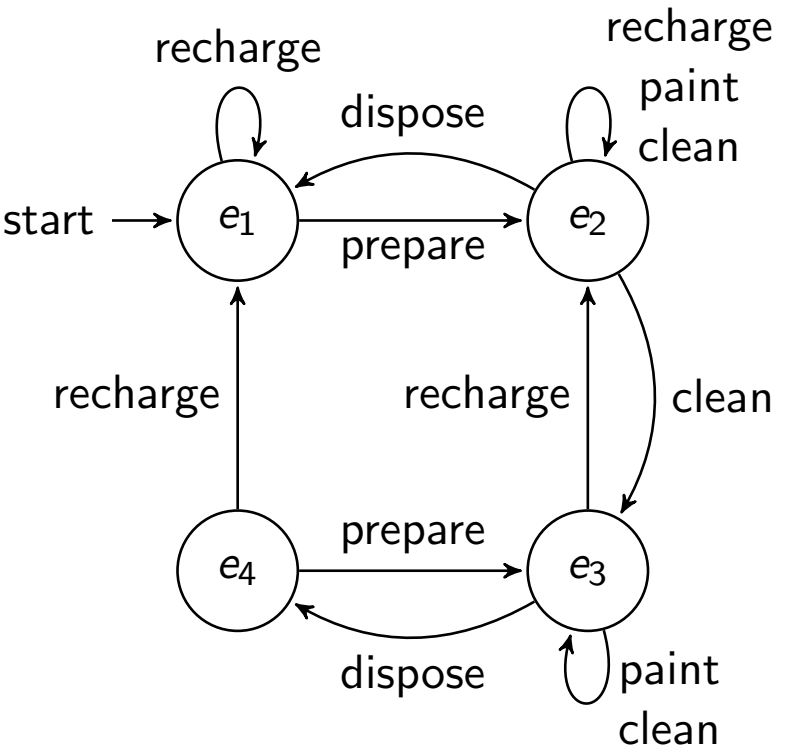


The Behavior Composition problem

Inputs:

- A library of available **behaviors**
- A **target** behavior
- An **environment***

The environment is the shared setting in which the available behaviors act and cooperate. In general, we have *incomplete information* about preconditions and effect of actions in the environment, thus we represent them with *non-deterministic* transitions systems:



The Behavior Composition problem

Inputs:

- A library of available **behaviors**
- A **target** behavior
- An **environment***

Output:

- A **controller**

The controller is a system component that is able to:

- Activate, stop or resume any of the behaviors
- Instruct any of the behaviors to execute an action

The controller realizes the target behavior by suitably delegating each action requested by the target behavior to one of the available behaviors.

Adopted Technology

- Unity Game Engine and its technical demo, “Angry Bots”
- C#
- LitJson
- Representational State Transfer (REST)
- XML/JSON
- Java
- Jersey
- JTLV
- The Composition class
- RestSharp

The Angry Bots patrolling domain

- Built upon Unity's Angry Bots technical demo
- Developed specifically to accommodate the Behavior Composition model in a video game-like scenario
- 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

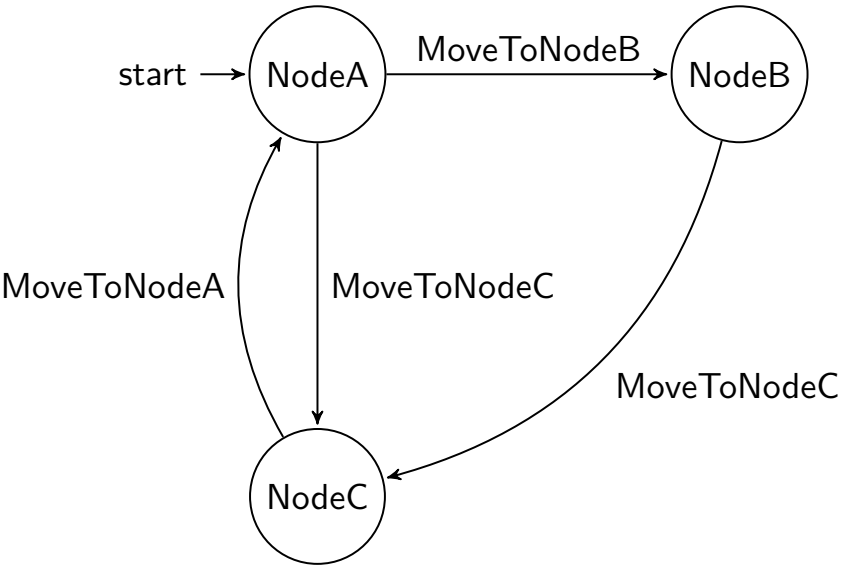
Relationship between the Behavior Composition framework and the Angry Bots patrolling domain:

- 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 Angry Bots patrolling domain

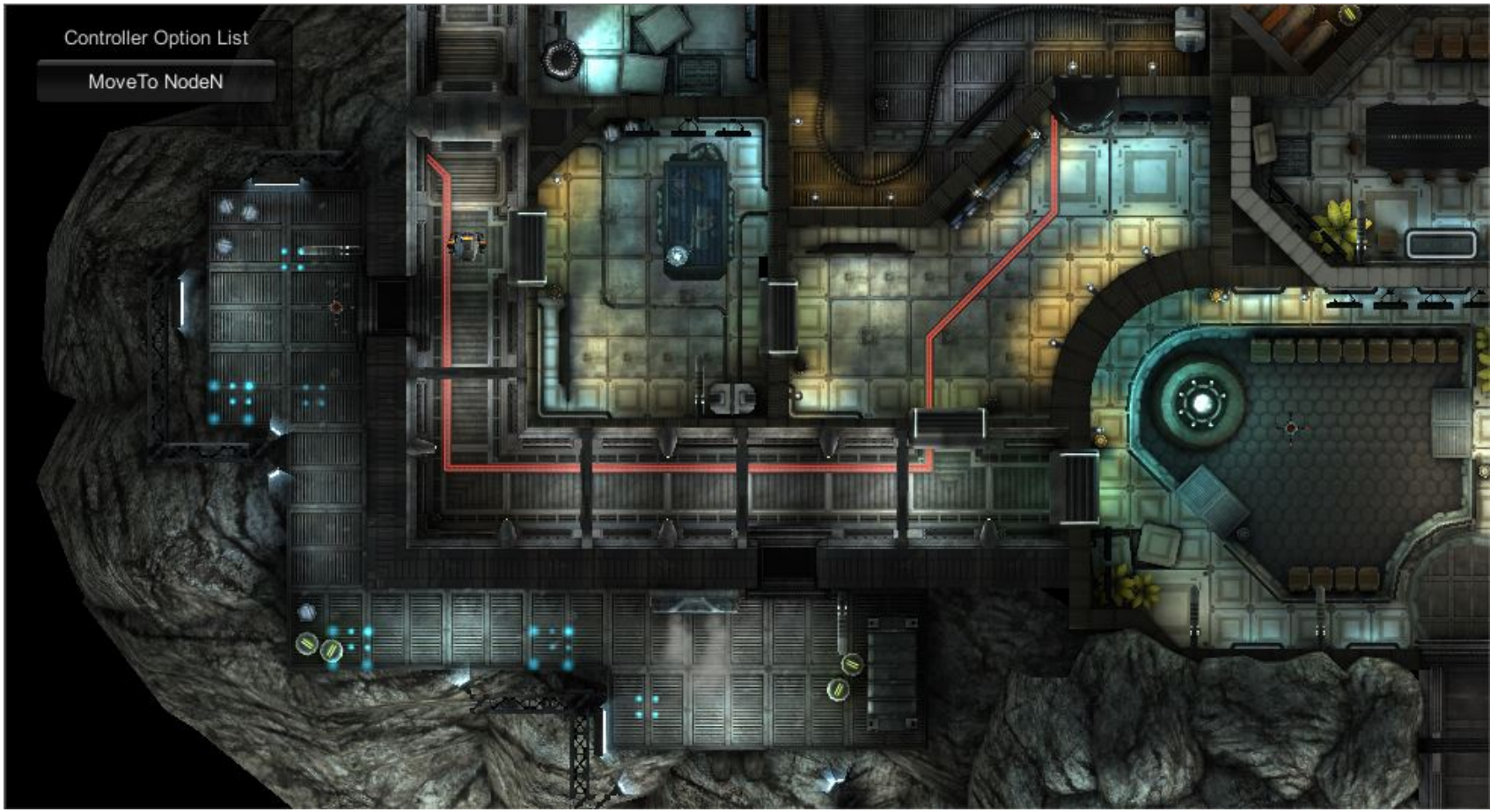
- The behaviors for the non-player characters, as well as the target behavior, are described using *finite state machines*, expressed in the **Trivial Graph Format**:

```
1 NodeA
2 NodeB
3 NodeC
#
1 2 MoveToNodeB
1 3 MoveToNodeC
2 3 MoveToNodeC
3 1 MoveToNodeA
```



- The framework is *interactive*: using a Graphical User Interface, the player takes control of the decisions in the target behavior

Screenshot of the Angry Bots patrolling domain



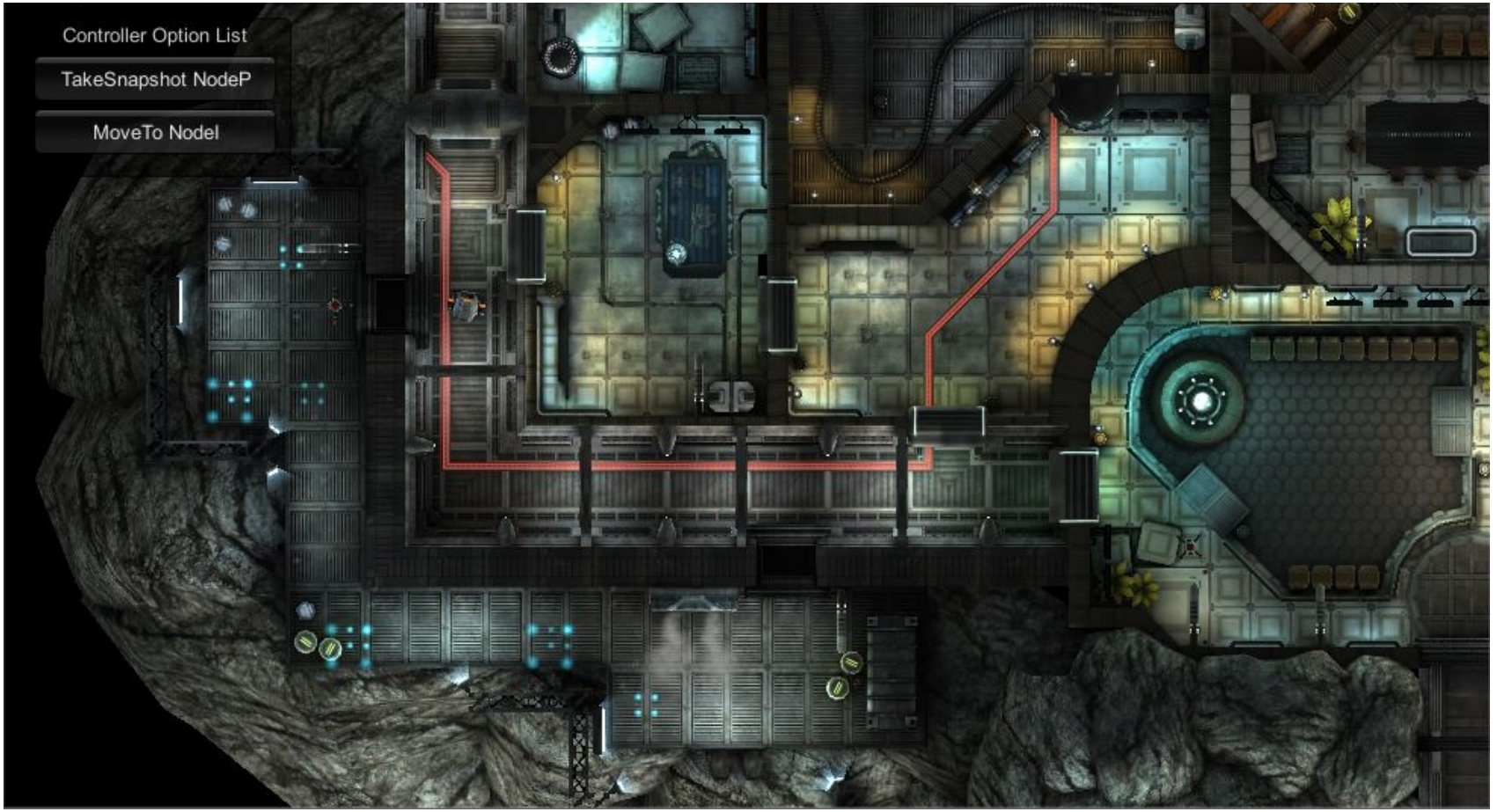
Behavior Composition using SM4LL

- The Smart Homes For All (SM4LL) Project uses the behavior composition concept for controlling services (doors, lights, etc.) automatically in a domestic environment

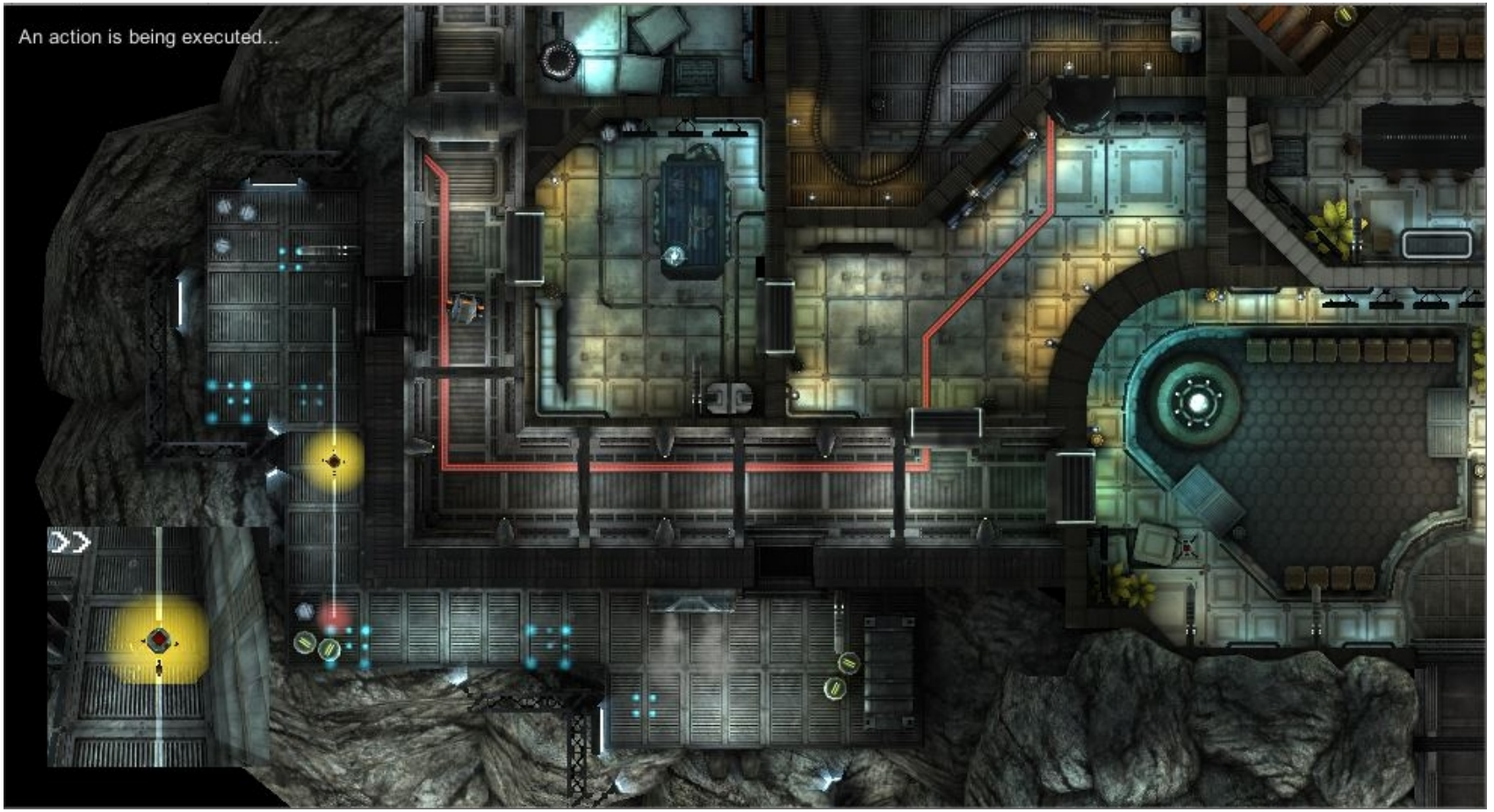
- We used their composition engine (called Off-line Synthesis Engine) for the first experiment with our framework

- Steps involved in getting a composition:
  1. Translate the behaviors of the non-player characters from the Trivial Graph Format to XML (eXtensible Markup Language) with the SM4LL schema definitions
  2. Group the XML files in a ZIP archive
  3. Submit the ZIP archive using the SM4LL web-based interface
  4. Wait for the calculation of the composition, and retrieve it as an XML file

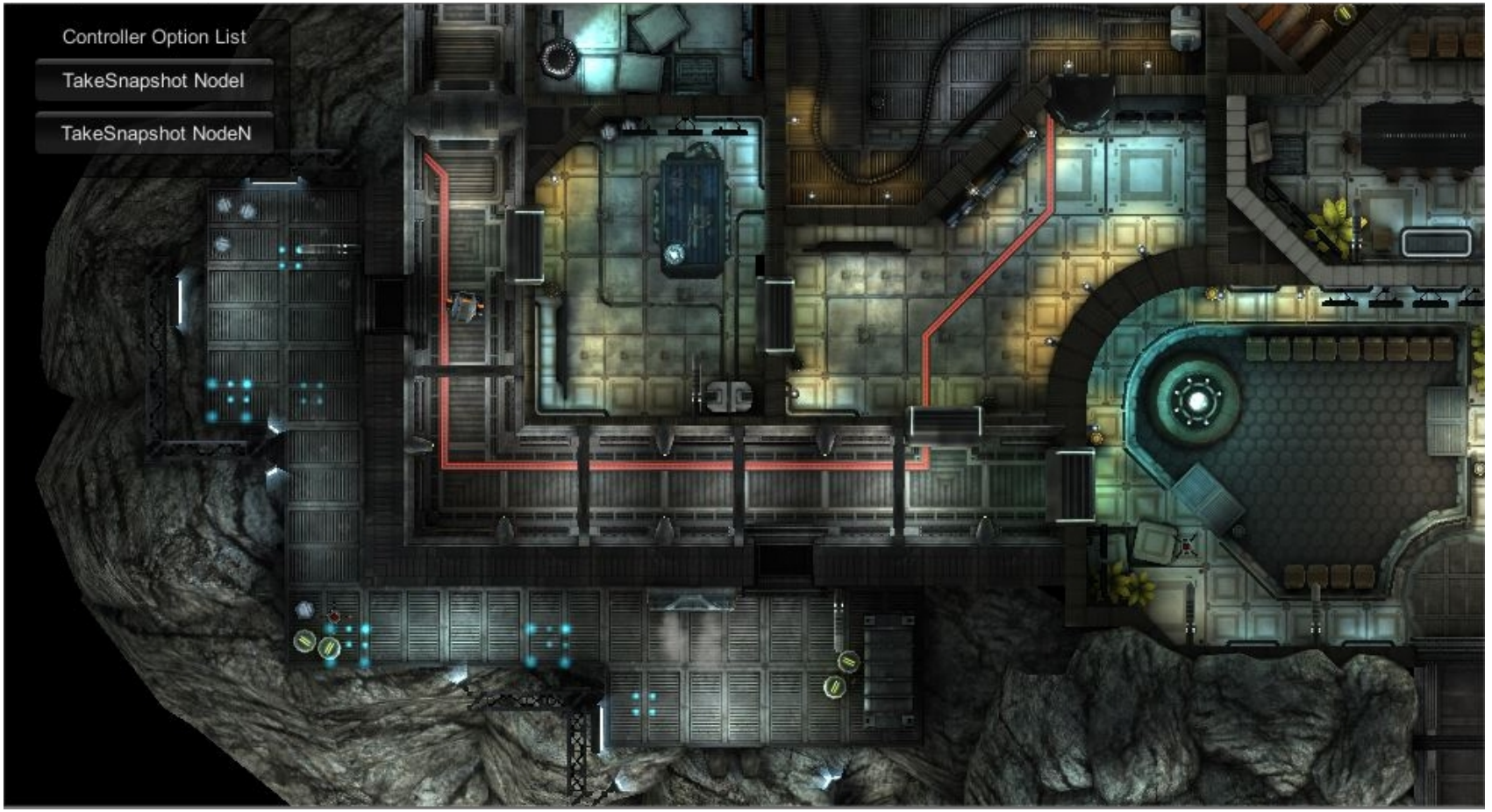
Screenshot of the Behavior Composition controller



Screenshot of the Behavior Composition controller



Screenshot of the Behavior Composition controller



The JaCO RESTful Web Service

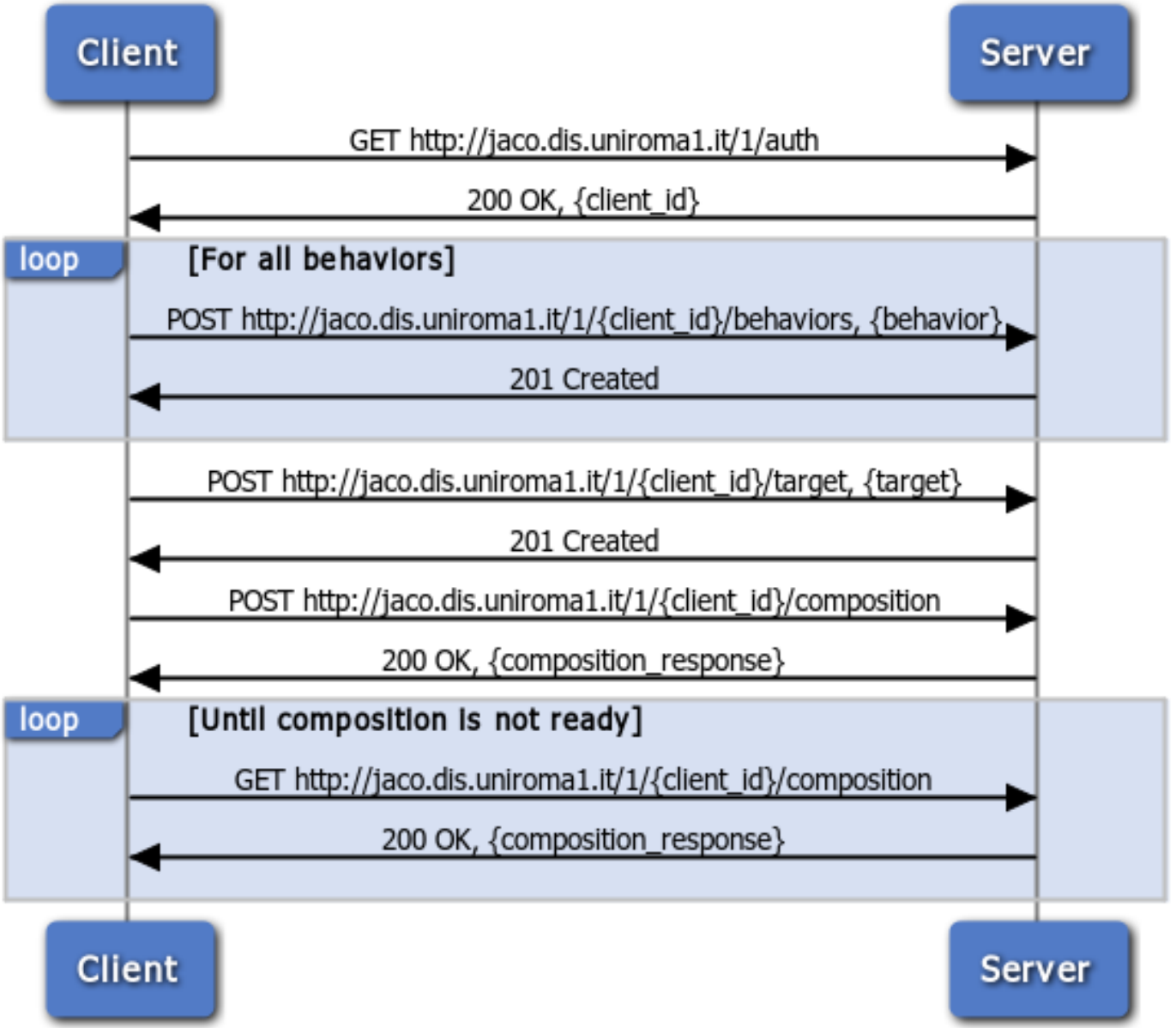
- Web Service written in Java, based on the Jersey library from Oracle, and following the REST principles
- Provides behavior composition-as-a-service
- The whole interaction between the client and the server is realized by sending and receiving HTTP messages
- Stands for **Ja**va-based **C**omposition-**O**riented Web Service

The JaCO RESTful 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



Usage scenario of the JaCO Web Service

```
C:\Users\Stefano Cianciulli\Downloads\curl>curl http://localhost:9799/jtlv/auth
--request GET
<?xml version="1.0" encoding="UTF-8" standalone="yes"?><client_id>bhcmngun4b2ffm
cg58h85emu0d</client_id>
```

```
C:\Users\Stefano Cianciulli\Downloads\curl>curl http://localhost:9799/jtlv/bhcmn
gun4b2ffmcg58h85emu0d/behaviors --request POST --header "Content-Type:text/xml"
--data @MyEnemyMech.xml
```

```
C:\Users\Stefano Cianciulli\Downloads\curl>curl http://localhost:9799/jtlv/bhcmn
gun4b2ffmcg58h85emu0d/target --request POST --header "Content-Type:text/xml" --d
ata @Target.xml
```

```
C:\Users\Stefano Cianciulli\Downloads\curl>curl http://localhost:9799/jtlv/bhcmn
gun4b2ffmcg58h85emu0d/composition --request POST
```

```
C:\Users\Stefano Cianciulli\Downloads\curl>curl http://localhost:9799/jtlv/bhcmn
gun4b2ffmcg58h85emu0d/composition --request GET
<?xml version="1.0" encoding="UTF-8" standalone="yes"?><compositionResponse><que
ue_number>0</queue_number><computedComposition><isRealizable>true</isRealizable>
<targetState stateName="a"><communityState><behaviorState service="MyEnemyMech"
state="NodeA"/><behaviorState service="MyEnemyMineBot" state="NodeH"/><behaviorS
tate service="MyEnemyMineBot1" state="NodeT"/></communityState><transitions><tra
nsition invoke="MyEnemyMech" action="MoveToNodeN"/></transitions></targetState><
targetState stateName="b"><communityState><behaviorState service="MyEnemyMech" s
tate="NodeN"/><behaviorState service="MyEnemyMineBot" state="NodeH"/><behaviorSt
ate service="MyEnemyMineBot1" state="NodeT"/></communityState><transitions><tran
sition invoke="MyEnemyMineBot1" action="MoveToNodeP"/></transitions></targetStat
e><targetState stateName="c"><communityState><behaviorState service="MyEnemyMech
" state="NodeN"/><behaviorState service="MyEnemyMineBot" state="NodeH"/><behavio
rState service="MyEnemyMineBot1" state="NodeP"/></communityState><transitions><t
```

The RESTfulClient class

- Class based on the RESTSharp library for integrating the interaction with the JaCO Web Service within the Unity game engine
- Methods defined:
  - `SendNonPlayerCharacter(string behaviorXmlString)`
  - `SendTargetBehavior(string targetXmlString)`
  - `RequestComposition()`
  - `GetComposition()`
- The string representation of the behaviors are retrieved by calling the `Serialize` method of the `FiniteStateMachine` class

RESTfulClient within Unity