Planning for Interactive Storytelling Processes

Stefano Cianciulli
Sapienza University of Rome
Rome, Italy
stefano.cianciulli@gmail.com

Stavros Vassos
Sapienza University of Rome,
Rome, Italy
vassos@dis.uniroma1.it

Abstract

In this paper we present some preliminary results experimenting with the AI method of behavior composition for the purpose of facilitating interactive storytelling in video games. The motivation is twofold: first, behavior composition is based on transition systems that are ubiquitous in video game development under the term finite state machines, and second, as the research community explores ways for a non-linear adaptive storyline in video games by means of automated planning and scheduling, the use of behavior composition may be able to offer added benefits by means of performing planning for a target desired process instead of a target desired state. We introduce JACO, a web service for behavior composition, and present a use case based on a simple conceptual example for interactive storytelling.

Introduction

In this paper we experiment with the AI method of behavior composition so as to facilitate interactive storytelling in video games. Behavior composition (De Giacomo, Patrizi, and Sardiña 2013) is concerned with *orchestrating* a set of available behaviors, each of which is expressed as a *transition system*, in order to accommodate a virtual target service also expressed as a transition system. The aim is to synthesize an orchestrator that is able to realize the target service by exploiting execution fragments of available services.

The motivation is twofold. First, transition systems are ubiquitous in video game development. Variants of transition systems, typically referred to as *finite state machines (FSMs)*, is one of the most widely used techniques for specifying the behavior of non-player characters (NPCs) in video games. This familiarity makes behavior composition well suited for orchestrating the behavior of NPCs. Second, as the research community explores ways for a non-linear adaptive and interactive storyline in video games by means of automated planning and scheduling, the use of behavior composition may be able to offer added benefits as transition systems essentially facilitate *planning for a target desired process* instead of a *target desired state*. For example, the target process may describe a recurring transportation activity connecting two areas, which may be realized by different available services, i.e., agents or devices.

In the setting we explore, each of the NPCs of the game may feature any preferred method for specifying and realizing their intended behavior, but we also assume that there is one additional interaction layer that specifies the *role of the NPC with respect to the storyline*. For each NPC a transition system or FSM is assumed that specifies which events in the storyline may be initiated and handled by the NPC and how they affect an internal state. For example, a particular NPC may be used to initiate a conversation with the player that reveals a clue or initiate a quest, but only if in the course of the game the player has not previously engaged in combat with the NPC in some previous encounter. Different states of the FSM may be used to represent the internal state of the NPC, and transitions may be used to encode available storyline interactions at each state. The set of these FSMs constitute the *available behaviors* for behavior composition.

As far as the intended storyline is concerned, a desired *target behavior* is constructed that describes how the events in the storyline may unfold. The desired target is not a fixed sequence of events, rather than another FSM that provides a high-level view of the process that the storyline should follow. Each state in the FSM corresponds to a *decision point* allowing a number of available storyline events to be invoked as transitions that lead to other states accordingly.

Finally, anticipating that service-oriented computing may become a useful paradigm for game development, we provide a RESTful web service for behavior composition, called JACO. The interaction with JACO is carried out by sending and receiving HTTP messages according to the REST principles. Our intention is to release JACO as a cloud-based tool that game developers can employ *offline* to compute an "AI orchestrator" for NPCs in a video game, which can then be used *online* to orchestrate NPCs according to the specified target and the choices of the "AI director".

A motivating example

We adopt a simplified game concept where the player embarks to a journey of becoming a mighty fighter or a powerful magician by pursuing various quests. A high-level view of this storyline is depicted as the desired target FSM in Figure 1. The storyline structure requires the player to begin his journey by completing a fixed quest, i.e., $quest0$. After

this initiation phase, he has the opportunity to pursue various quests, i.e., quest1–quest7, which influence the path he is taking toward becoming a fighter or a magician. For example, different sequences of quests may result to the storyline FSM being in state "Fighter", "Magician", or "Main". Note that only from the states "Fighter" and "Magician" can the storyline evolve to the ending through an appropriate quest.

The target FSM does not prescribe the desired sequence of events; it only specifies a *process* that the storyline should comply with. A so-called *AI director* component could use this as the basis for presenting available quests to the player to choose in order to progress the story. A more sophisticated AI director could look into the available options and choose to progress the story using the one that would be considered more fun for the player taking into account other information. In any case, each state in the target FSM is a *decision point* which specifies the possible ways to proceed. The actual realization of the decision though is to be performed by one of the available NPCs, also expressed as FSMs.

In our simple example there are six NPCs that the player may interact with for the purposes of achieving quests. Each NPC may be involved in more than one quest affecting the evolution of the storyline with respect to the target FSM of Figure 1, but also affecting the disposition of the NPC in a positive or negative way. Depending then its disposition, a different set of quests may be facilitated by the NPC in the course of the game. For example, Character1 may be used to facilitate quest3 and quest4 but not both. Similarly, Character2 may facilitate quests 3,5, and 6, but executing quest6 may get him into "Negative" state in which only quest6 could be re-invoked. Note also that the same quest may have a positive effect for one character but a negative effect on another, e.g., quest3 for characters 2 and 3. Also, for simplicity we have allowed quests to reoccur in this specification – in practice this may be reasonable only for some quests.

A problem that arises is that it is not easy to guarantee that the available FSMs corresponding to NPCs can *realize all possible runs* of the specified target FSM. For example consider the following scenario:

- Character3 facilitates quest0: this causes the storyline to evolve from "Beginning" to "Main" state, and Character3 to change its internal state from "Neutral" to "Positive";
- Character4 facilitates quest1: this causes the storyline to evolve from "Main" to "Fighter" state, and Character4 to change its internal state from "Neutral" to "Negative";
- Character4 facilitates quest7: this causes the storyline to go back to "Main" state, and Character4 to change its internal state from "Negative" to "Positive".

At this point, according to the target FSM the player should still be able to get involved with quest1 or quest2, but the only character that is capable of facilitating it, i.e., Character4, is in "Positive" state, from which he is not able to facilitate these quests.

The method of behavior composition that we described in the introduction is able to automatically construct a *global strategy* that specifies how each of the transitions of the target FSM should be delegated to available FSMs in order to avoid such situations *for any possible run of the target transition system*. In the next section we present JACO, a web service that provides this functionality.

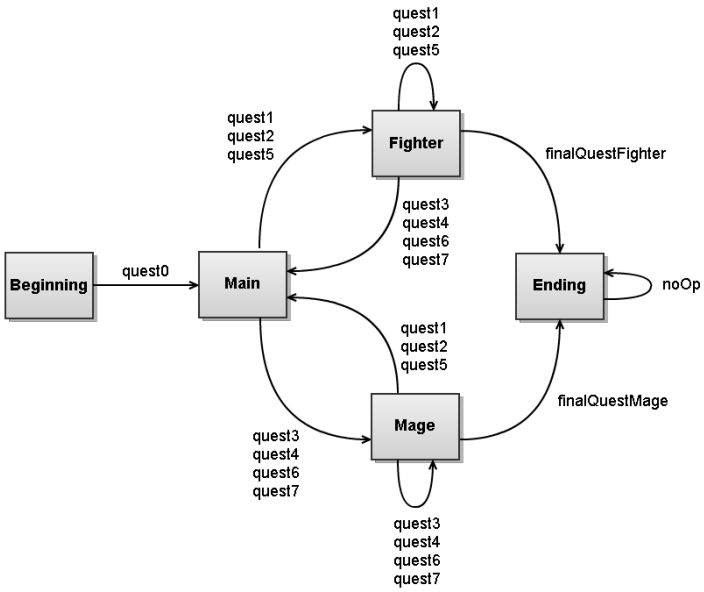


Figure 1: Storyline as a target FSM

Behavior composition with JACO

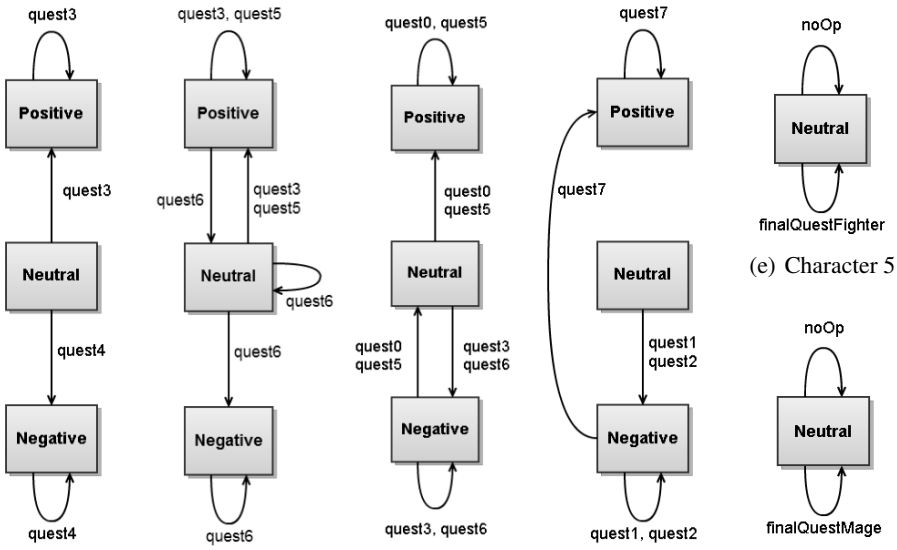
JACO is a web service for behavior composition following the REST software architecture. For a formal definition of behavior composition the reader is referred to (De Giacomo, Patrizi, and Sardiña 2013). The interaction with JACO is done by sending and receiving HTTP messages that handle information and requests. JACO's main computational component is built on top of *JTLV*[1],i.e., a Java implementation of the Temporal Logic Verifier (Pnueli and Shahar 1996).

JACO performs the following computation: given (*i*) a set of *available behaviors* as possibly nondeterministic FSMs and (*ii*) a *target behavior* as a deterministic FSM that is to be realized by combining execution fragments of the available behaviors, JACO provides a *composition* that specifies how the target can be realized. The composition can be used as a *look-up table* specifying for every collective state of the available behaviors and the target, which of the behavior can be used to realize the transitions supported by the target.

The JACO API identifies five endpoints with which the user can interact using HTTP verbs, such as GET and POST, in order to send or receive information expressed in XML:

- **/auth**: provides the user with a unique identifier **client_id** to be used with all requests to JACO as follows;
- **/{client_id}/behaviors**: allows the user to retrieve a list of available behaviors that have been submitted to the server by issuing an HTTP GET request, or add a new behavior with an HTTP POST request;
- **/{client_id}/behavior/{behavior_id}**: allows the user to perform the usual CRUD operations (Create, Read, Update, and Delete) on the behavior identified by **behavior_id** using HTTP GET, POST, PUT, DELETE;

[1] http://jtlv.ysaar.net/



(a) Character 1 (b) Character 2 (c) Character 3 (d) Character 4 (e) Character 5 (f) Character 6

Figure 2: The finite state machines of the six characters of the domain

- **/{client_id}/target**: allows the user to perform the CRUD operations to set, update or delete the desired target for behavior composition;
- **/{client_id}/composition**: allows the user to request the computation of the specified behavior composition problem by issuing a POST request, and obtain the resulting composition (or the status of the operation if it is still processing or in queue) using a GET HTTP request.

In a typical JACO usage scenario the user starts by getting a **client_id**, and then using appropriate POST requests specifies available behaviors and the desired target behavior, as well as initiates the computation of the behavior composition. Then the user goes on a loop of appropriate GET requests with which the status of the composition is retrieved, until the actual composition is returned as output. More information about the API and the XML data that is sent and received by the user can be found at `http://jaco.dis.uniroma1.it/`.

We now proceed to show how JACO can be employed to orchestrate the NPCs of Figure 2 in order to guarantee the realization of the target FSM of Figure 1.

JACO in action

Each of the NPCs of Figure 2 is specified as an available behavior using XML. The XML representation is very simple including a name and a finite state machine specified using nodes and transitions. For example, the following XML listing is used for Character1:

```
<behavior>
  <name>Character1</name>
  <finiteStateMachine>
    <state node="neutral">
      <transition action="quest3">
        <target>positive</target>
      </transition>
      <transition action="quest4">
        <target>negative</target>
      </transition>
    </state>
    <state node="positive">
      <transition action="quest3">
        <target>positive</target>
      </transition>
    </state>
    <state node="negative">
      <transition action="quest4">
        <target>negative</target>
      </transition>
    </state>
  </finiteStateMachine>
</behavior>
```

The target behavior is also represented using the same tags. The following is an excerpt used for the target of Figure 1:

```
<behavior>
  <name>Target</name>
  <finiteStateMachine>
    <state node="beginning">
      <transition action="quest0">
        <target>main</target>
      </transition>
    </state>
```

Following the usage scenario of JACO we get a client id and post the available behaviors and the target behavior one by one. We then request a composition that would provide an orchestrating strategy. If one such strategy exists, it will ensure that we can always continue progressing our storyline following the options formalized in the target FSM. Moreover for every available transition in the target FSM (and every corresponding possible state of all available FSMs), it will instruct exactly which of the available FSMs we should choose to realize the transition in order to ensure this.

When we get back an answer from JACO, the result is that such a composition is not possible. This means that some problematic runs for the target FSM (like the one we identified involving characters 3 and 4, and quests 0,1,2, and 7, in the section of the motivating example) cannot be avoided

by delegating the quests to characters in a different way. Essentially, *there exists some run for the target FSM* such that there is *no way to realize using the available FSMs*.

As a piece of information this is important to know but it is not very helpful for the purpose of specifying and executing an interactive storytelling experience as we intended. Nonetheless, in order to investigate which are all the problematic cases that lead to a deadlock, we can proceed as follows. We specify a simple "all-purpose" character that can facilitate all available quests always staying in the same "Neutral" state, and post it as an additional available behavior. Then we request a composition again. As expected, this time a composition is possible, but what is more interesting is that the details of the composition[2] point out the problematic cases. The composition shows at each step information about which of the available FSMs could facilitate a transition of the FSM based on the current state of the available FSMs. The cases then where only the "all-purpose" character comes out as an option are the ones that would originally lead to a deadlock. Essentially, this provides a simple way to "debug" storyline processes and behaviors at design time.

Related work

Our approach is similar in spirit to many other approaches in the literature that are based on automated planning, including STRIPS and HTN planning, for example the system I-Storytelling (Cavazza, Charles, and Mead 2002), GADIN (Barber and Kudenko 2009), and MIST (Paul et al. 2010) as well as the work on the framework Mimesis (Young 2001) and Zócalo (Young et al. 2011). Nonetheless, the methodology of behavior composition is different from planning both in conceptual and technical terms as we explain next.

Firstly, the target behavior is not a specification of a goal situation to reach but, rather, a description of a set of *routines* one would like to be able to carry on *at runtime*. Moreover, such routines cannot be seen as (classical or nondeterministic) plans, either, in that they do not prescribe the actions to execute, but leave the choice to the executor. Further, they may contain loops, which are typically ruled out in planning. From this perspective, target behaviors are more similar to IndiGolog programs (De Giacomo et al. 2009), i.e., high-level procedures definable on top of planning domains, for which one is typically interested to find an executable realization at runtime.

Secondly, in behavior composition, actions are not the subject of a planning task. Indeed, the controller does not select the actions to execute; instead it returns the index of the behavior that should execute the action selected by the AI director. In this sense, actions constitute the input, not the output, of the reasoning task, but in a way that takes into account *all possible narrative trajectories*. From a more formal perspective, we observe that both behavior composition and *conditional planning* are EXPTIME-complete problems (De Giacomo, Patrizi, and Sardiña 2013; Littman 1997), thus some way of reducing composition to (nondeterministic) planning must exist. Nonetheless, how this can actually be done is not as straightforward as one might expect, as shown by the above considerations.

Finally, our implementation of the behavior composition engine as the web-service Jaco is similar to the client-server based approach that is adopted in Mimesis and Zócalo. In fact as JACO is built as a pure behavior composition engine that can be accessed via a REST API, one interesting direction for future work is to explore how it can be used it as a service in such frameworks in order to provide high-level orchestration of characters, either as an alternative or in pair with the embedded narrative planner.

Conclusions

In this paper we present some preliminary results experimenting with the AI method of behavior composition for the purpose of facilitating interactive storytelling in video games. We motivate the use of this method with a simple conceptual example that would require the orchestration of various non-player characters in order to facilitate different parts of the story. Anticipating that service-oriented computing may become a useful paradigm for this type of aspects of game development, we provide a RESTful web service for behavior composition, called JACO, and we use it to provide solutions for the motivating example. Our preliminary results show that such a service can be useful for use cases similar to our motivating example, and our current work focuses on identifying scenarios coming from commercial video games to validate our approach.

References

Barber, H., and Kudenko, D. 2009. Generation of adaptive Dilemma-Based interactive narratives. *Computational Intelligence and AI in Games, IEEE Transactions on* 1(4):309–326.

Cavazza, M.; Charles, F.; and Mead, S. J. 2002. Character-Based interactive storytelling. *IEEE Intelligent Systems* 17(4):17–24.

De Giacomo, G.; Lespérance, Y.; Levesque, H. J.; and Sardina, S. 2009. IndiGolog: A High-Level programming language for embedded reasoning agents. In *Multi-Agent Programming: Languages, Tools and Applications*. 31–72.

De Giacomo, G.; Patrizi, F.; and Sardiña, S. 2013. Automatic Behavior Composition Synthesis. *Artif. Intell.* 196:106–142.

Littman, M. L. 1997. Probabilistic Propositional Planning: Representations and Complexity. In *Proc. of AAAI 97 and IAAI 97*, 748–754.

Paul, R.; Charles, D.; McNeill, M.; and McSherry, D. 2010. MIST: An interactive storytelling system with variable character behavior. In *Interactive Storytelling*, volume 6432 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 4–15.

Pnueli, A., and Shahar, E. 1996. A platform for combining deductive with algorithmic verification. In *Proceedings of the Eighth International Conference on Computer Aided Verification*.

Young, R. M.; Thomas, J.; Bevan, C.; and Cassel, B. A. 2011. Zócalo: A service-oriented architecture facilitating sharing of computational resources in interactive narrative research. In *Working Notes of the Workshop on Sharing Interactive Digital Storytelling Technologies at ICIDS*.

Young, R. M. 2001. An overview of the mimesis architecture: Integrating intelligent narrative control into an existing gaming environment. In *Working Notes of the AAAI Spring Symposium on Artificial Intelligence and Interactive Entertainment*.

[2] XML files are available at JACO website.