Virtual agents for the production of linear animations

Rossana Damiano[a], Vincenzo Lombardo[a], Fabrizio Nunnari[1]

[a] *CIRMA - Università di Torino*
[b] *DFKI*

Abstract

In the last decade, a number of techniques from the new media practices have contributed to innovate the traditional production of entertainment, through the modularization and the automation of a number of phases. This paper proposes a novel approach to the automatic generation of character animations that draws inspiration from the techniques for the construction of the virtual agents. The pipeline for the production of animated scenes is based on the mapping between the authorial description of characters' behavior and the actual animation data. The application context is the production of linear (non interactive) animations. Given the specification of a set of high level goals, the implemented system generates the animation through the generation of a sequence of actions, the translation of actions into animation commands, the display of an animated scene through a 3D graphic engine. The pipeline and the system are validated onto the production of a short animated movie, with the participation of a commercial company.

Keywords:
virtual agents, character animation, animation languages, HTN planning

1. Introduction

In the last decade, a number of techniques from the new media practices have contributed to the innovation of the traditional production of entertainment, introducing modularization and automation in a number of phases of the production process [1]. In this paper, we investigate on the generation of a character's animations from a high level description of its behavior.

The production pipeline of the character animation [2], with particular reference to 3D animation, starts from the screenplay written by an author and is accomplished through a a series of phases. The director, together with

the storyboard artists, creates an animated storyboard (also called "animatics"), in which the scenes described in the screenplay are visualized through a sequence of drawings, usually timed according to some audio track; the graphic artists create backgrounds, objects, characters, and assemble them into scenes; the animators (possibly coordinated by an animation director) animate the characters, breaking down their actions and editing the animation curves through the use of sophisticate graphic editors (that hide the math and the numbers behind 3D computation). Finally, scenes are edited following the animatics, with variations in timing and rhythm, which are refined following music beats and cinematographic principles; all the scenes are assembled together into a linear animated movie.

For the sake of expressivity, the animation curves generally expose a high number of controls to the animators. A simple gesture, like grabbing an object, involves more joints than those located in arm and fingers, since the whole body must follow to reflect stretching and change of balance [2]. For modern, full-featured characters, more than one hundred animation curves can be associated to one animation clip; a "simple" skeleton is made of about 100 bones, with 3 degrees of freedom for each bone.

The automation of character animation raises issues at different levels of specification. At low level, where human animators can control single animation curves, the techniques for the automatization of the animation process include procedural techniques, such as Inverse Kinematics solvers, which ease the translation of hands and feet in space, accounting for whole chains of body segments [3, chapter 5.3], and physical simulation, which achieves realism in balancing situations with complex body models and applied forces (e.g., gravity) [4]. However, nowadays, only a limited set of animations can be accomplished fully procedurally (e.g., automatic eye gaze), while maintaining a highly expressive animation output.

At high level, the problem of bridging the gap between the story – conceived of by an author and possibly including an interactive script – and the generation of the animation, has been addressed by the research in virtual agents. The integration of these two levels is particularly complex in interactive applications, which employ autonomous virtual agents to cope with the range of behaviors required by the interactivity. Autonomous agents are widespread in a range of applicative domains that span from cultural heritage [5, 6, 7] and education [8, 9] to interactive drama [10, 11] and conversational agents [12, 13]. Authoring the behavior of such agents is an interdisciplinary task, that requires the cooperation of several disciplines and

professional roles, including knowledge and software engineers, programmers, graphic artists and animators [14]. The "procedural author" [15] specifies the basic "bricks" that constitute the behavior of the character in the range of situations encompassed by the interactive story. Concerning the animation of the character, we can identify two relevant roles: a 3D animator, who animates the behavioral "bricks" (clips representing actions, such as a walk cycle, and poses for well distinctive action strokes, such as the expression of surprise), and the 3D programmer, who works on the procedural generation of the animation bricks (e.g. gaze direction) and their automatic composition.

The goal of the work presented in this paper is to apply the techniques derived from the field of virtual agents to the production of linear (i.e. non interactive) animation. In particular, we rely on the techniques of agent deliberation to interpret the directions contained in the screenplay, generating a sequence of actions from a high level specification of the character's behavior; then, actions are translated into animated scenes through a real time graphic engine. The pipeline and the system introduced by this paper is aimed at saving time in both the conception and realization of animation: the high–level behavior of the agent, stated in the screenplay (and usually reinterpreted or detailed out by the director or the animators in the traditional pipeline), is given as input to the system as a set of generic tasks, without specifying the details of how they will be accomplished in some particular scene/situation; the system translates the characters' tasks into practical actions and then into the actual animations for a given situation. This process can support the work of the director and the animators, by providing a first approximation of the final result, or can be directly employed to the generation of the character's animated behavior, when the quality required for the animation is not high.

Given the applicative context of the linear animation, the novel contributions of the paper are:

- a production pipeline for authoring animated characters from high level behavior specifications;
- a reference architecture for generating the animation from these specifications;
- a declarative language for mapping the character's behavior onto animations.

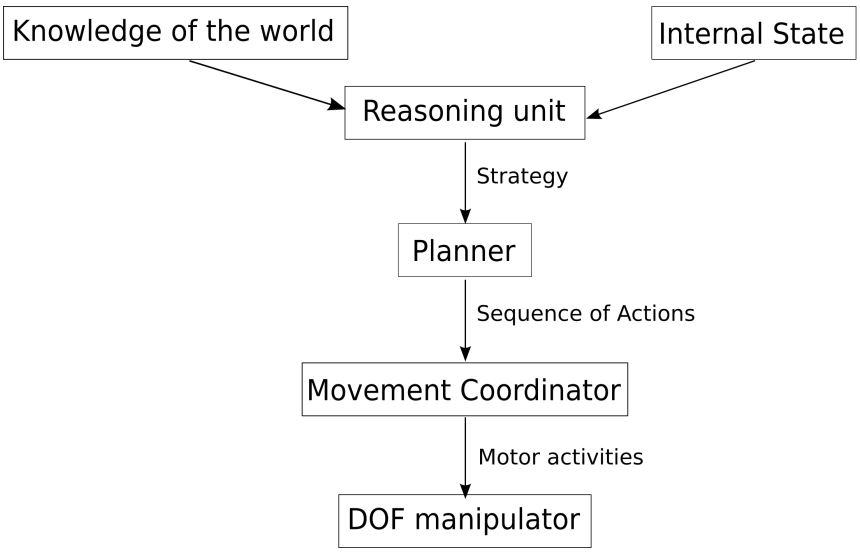

Figure 1: Levels of behavior of a virtual agent (adapted from [3]).

The whole approach, called AnimaTricks, was validated on a pilot linear animation (a short movie with a narrative content), with the participation of a production company.[1] The generated animation were evaluated by a panel of producers and consultants.

The structure of the paper is the following. After surveying the state of the art of the techniques for animated agents (Section 2), we describe (Section 3) the AnimaTricks pipeline, system architecture and the language we used to specify the agent's behavior and the action-animation mapping. Finally, we describe the case study and comment on the results (Section 4). Conclusions and future work end the paper.

2. Animated characters

Most of the research on animated agents was stimulated by the design and implementation of artificial characters and intelligent virtual agents in interactive applications.

The schema in Figure 1 depicts the levels of behavior of a virtual agent, going from the most abstract to the most concrete activities. The Reasoning

[1]The project AnimaTricks was funded by Regione Piemonte, settore Cultura, 2009.

unit, taking into account the internal state of the agent (current goal, action history, the perception of the world) and the world knowledge, deliberates a strategy that is passed to the Planner (usually an updated goal). The Planner, which relies upon a library of plans associated to goals, calculates the sequence of actions. The actions are coordinated in their executions (Movement Coordinator), while the actual animation is operated by the DOF manipulator, which manipulates the degrees of freedom on the character's joints and blend-shapes.

The first two units constitute the deliberative, AI–based component of the virtual agent. In our application context, representation and reasoning are delegated to the author and we assume that they are distilled into the authorial directives in the form of task specifications. As for the planner, virtual agent architectures are based on multiple approaches, ranging from HTN planning (Hierarchical Task Network) [16, 17] to heuristic search planning [11]. A debated issue is the coupling of the planner with the graphic engine that implements the two lowest levels, which are responsible for the generation of the motor activities. Some approaches, such as [11], have explored the coupling with external graphic engines, such as Unreal[2] or Unity[3]: this scheme guarantees the real time reactivity of the character, but opens the issue of the coordination between the behavioural drive and the graphic engine [18]. Other approaches have pursued the complete integration between the deliberation and the animation component, leading to the development of layered architectures [19, 20, 21], that integrate the deliberative and the animation levels through the design of animation languages that encode the character's behavior in terms of animation primitives.

Three observations make layered architectures the appropriate candidates for applying modularization and automation to the production of linear animation – where interactivity and real time response are not required. First, in order for the architecture to be applicable in a pipeline where some figures have no programming skills, it must support a declarative approach to the definition of actions, so as to facilitate the task for the 3D animator, who is responsible for the action breakdown into basic animations and their realization as animation clips, possibly split into different tracks for the various body parts; second, the system must enable the definition of parameters

[2]http://www.unrealengine.com/ [last visited on 8 May 2013]
[3]http://unity3d.com/ [last visited on 8 May 2013]

for the declared actions: this greatly improves the modularization of animations, reducing the number of definitions for similar actions in slightly different contexts, and supporting the re-usability of work across different scenes and projects; third, the engine must provide an animation system that works both data-driven and procedurally. On the one hand, the 3D animator must be allowed to provide complex expressive animations (that cannot be realized through the automatic merge of basic motor primitives); on the other, the procedural generation and the alteration of data-driven movements are a key feature to support the parametrization. Hence, the system should give 3D programmers the means to alter, parametrize and mix (blending and layering, see below) human-made poses and animations.

Another concern of our architecture is about time management. The deliberative process outputs a sequence of actions, to be performed in the specified order. No timing information is specified. We believe that perfect timing specification is not a crucial issue in action definition: it yields to a repetitive iterative authoring process to fine-tune the desired result and poses limits to a further (future) extension of the system to more interactive environments. Hence, in our system time is neither exposed as parameter nor used in the action execution. Differently, we specify animation "speeds", for which it is easier to identify default values (e.g., walk speed), and whose duration depends on the specific scene state during execution (e.g., according to actual path length).

In particular, in our work, we are interested in defining a declarative language (from now on, an Action to Animation definition language, or A2A) to define parameterized actions through the use of motor activities comprising both data-driven and procedural animations. So, the movement coordinator plays the role of the interpreter of the A2A definitions, while motor activities are hard-coded in an animation engine (DOF manipulator). Many character animation engines have been developed in the past (such as the C4 system [20]), which provide all the functionalities to drive autonomous characters. However, they do not expose a declarative language to define new actions and animations, making it difficult to adapt the system to new environments.

A number of declarative languages have been proposed since the dawning of character animation, allowing the definition of actions that can be processed by a Movement Coordinator. The pioneering PAR (Parameterized Action Representation) language [22] is a template-based representation of actions designed to program animated agents, bridging the gap between nat-

ural language instructions and the actual animations. A PAR definition, that includes the agent of the action, the relevant objects, the path to follow, location, manner, the purpose of a particular action, and the logical preconditions and effects, provides the means to define the parameters and mixes the logical information with the description of its animation. In our architecture the former aspect is handled within the specification of the behavioral plans, and the latter is left to the animation language: the production pipeline identifies an AI engineer and a 3D programmer for these two levels respectively.

BML (Behavior Markup Language, one of the most documented and solidly implemented languages) [23] and BEAT (Behavior Expression Animation Toolkit) [24] are animation languages conceived to animate intelligent conversational virtual agents, geared to describe the multimodal communicative behavior of an agent. They focus on the ability to synchronize human gesture with speech. Whilst data-driven animation is supported, these languages are specific to human gesture for conversation, not for generic human movement. Animation of custom body parts, as well as the control of non-humans, is not supported; additionally, timing specification is a necessary requirement of the language, and so, according to our requirements above, is not appropriate for the parameterization of linear animations.

The Improv system supports the description of both animation and behavior of actors on stage [19]. Its oversimplified behavioral sub-system is based on simple sequential script approach (much poorer with respect to the HTN approach of this paper). The animation layer, and its related language, despite not exposing a parameterization mechanism, and requiring explicit time information, share some concepts with our approach. The Improv animation language supports the definitions of actions by imposing a change of value for a set of degrees of freedom (DOFs) in time. In modern terminology, a DOF can be either the rotation of a bone around a reference axis (x, y or z) in skeletal animation or the weight of a blend pose for blend–shape animation. We took inspiration from Improv in its ability to let the author compose actions. First, the author can specify sequences of actions; second, by defining "priority groups", the author specifies the parallelism of actions, giving the author the means to specify how to handle overlapping DOFs variation in time. In our animation language we further extended the sequential/parallel-with-priority approach by allowing the definition of nested, sequences and recursive parallel groups. Moreover, we include as DOFs the global position and rotation of the character in space.

EMBRScript is the animation language of the EMBR animation engine

[25]. It allows the description of animated gestures through the composition of both data-driven and procedural animations. It was used to implement a BML realizer, providing all the features needed to accomplish the conversational animation gestures. However, the language is bound to the human movements and a finite preset of body parts and autonomous behaviors. The disadvantage consists in having a granularity in handling layered animations to the level of body parts, instead of single DOFs (such in Improv and our system). This limited granularity problem arises since hand-made animation tend to be made of animation curves scattered throughout the whole body skeleton. More advanced procedural features, like automatic eye-gaze and settling of the body according to its balance, can be inserted into our system as dedicated animation tracks. Concerning balance, differently from EMBRScript, where settlement is used to explicitly move the character to simulate change of balance, in our system a physical simulation would be used to re-settle a posture that can has been altered out of its believability. Moreover, EMBR requires the specification of many temporal constraints. This can be acceptable within the BML architecture, where such constraints are meant to be generated by an automatic resolver, but it can be a hard job for a human author in the pipeline. Finally, the language does not support the exposition of parameters in newly defined animations.

In the following, we present the AnimaTricks system, including the pipeline and the architecture, with a particular reference onto the representation language, all specialized for the context of the linear production of animations.

3. The AnimaTricks System

The architecture of the system includes three main components (Figure 2): the Planner, the A2A Executor and the Animation engine. Each module in the architecture is related to a specific knowledge base (the cylinders in Fig. 2). For the Planner, the knowledge base consists of the plan library. For the A2A Executor, the knowledge base consists in the catalogue of actions. The Animation Engine relies on the repository of animation data.

The AnimaTricks System includes an *offline Authoring Pipeline*, where contents (characters' behavior specifications and animation data) are created, and an *online System Architecture*, where the system is run to generate the character animation.

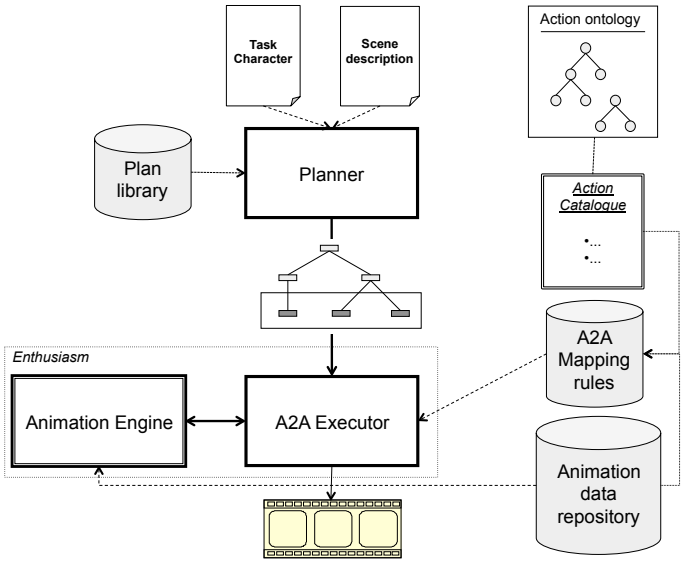

Figure 2: A graphical representation of the reference architecture. Solid lines represent the control flow, dashed lines the data flow.

3.1. Authoring Pipeline

The offline phase includes the **Behavior definition**, in which the writer, with the help of the AI engineer, encodes a set of character's behaviors into the format required by the planner (Plan Library), and the **Catalogue creation**, in which the action primitives contained in the Plan Library are translated into the animation language (creating manual animations when required), labeled and stored in the Action Catalogue. In this pipeline, the animator is required to adopt a 'new media language' approach [1], by specifying configurations that will be reached through automatic procedures (the *automation* principle) and by animating re-usable modules (the *modularity* principle) that are dynamically employed to create new sequences.

Behavior definition. Given the input provided by the writer, the AI engineer designs and implements a library of plans that encodes the character's behavior and tests them in a range of representative of situations. Plans are evaluated by the writer to make sure that the generated behavior is acceptable, and iteratively modified if necessary. In AnimaTricks, plans are encoded in the HTN paradigm, i.e., by describing complex, high–level

tasks as sequences of simpler, lower level tasks.

Catalogue creation. Given the rigged 3D model of the character, created according to the specifications provided by the writer, the animator's work consists in cooperating with the 3D programmer to implement the actions contained in the Behavior Library. Each action that is already stored in the Action Catalogue (from a previous project) is a candidate for reuse. Otherwise, if the animation of the action is produced from scratch, the animator and the 3D programmer analyze the structure of the action and break it down into its motor components; the 3D programmer evaluates if the action can be procedurally generated through an animation language expression (ActionToAnimation - A2A - mapping). Even if the animation is procedurally generated, it may be necessary (or more convenient for quality requirements) that one or more parts of the action are manually edited. For example, for some action types (such as facial takes of joy or surprise), it is advisable to manually produce the pose that marks the culmination of the action. Finally, the new entry is added to the catalogue, accompanied by a textual label and stored in the repository.

In the online phase, the system generates the animated behavior of the character given a specification of the character's high level tasks. This phase is similar to the traditional production methodology, where the writer provides a set of directives containing a high level description of the scene, including the behavior of extra characters (for example, "doing office work"), further refined by the director through the exact staging of the scene elements (characters and objects), i.e., a detailed the description of their positions, activities and trajectories, accompanied with exact timings. In AnimaTricks, these directives are translated into a formal specification of high-level tasks that the system interprets to generate the animation of the characters through planning and procedural animation.

3.2. System Architecture

Planning. The input to the Planner consists of a description of the scene and a set of tasks. In the current implementation of the system, we use the JSHOP2 planning system [26, 27], so actions are encoded in JSHOP2 language. For example, here below is the description (called *method*) of the task consisting of taking an object.[4] The method (`take`, line 1) features two

[4]The example, taken from the implemented system, is encoded in JSHOP2 plan de-

parameters, an agent (`?agent`) and an object (`?object`), and encompasses two alternatives: if the agent is in the same location as the object (line 5), it simply grabs it (line 7); otherwise (lines 10–12), the agent moves to the object location and then grabs it (line 14). The actions of going and grabbing are represented, respectively, by the methods `go` then `grab`.

```
1  (:method (take ?agent ?object)
2
3  ;agent and object are in same location
4  ((agent ?agent) (object ?object) (location ?loc)
5   (at ?agent ?loc)(at ?object ?loc))
6  ;simply grab the object
7  ((grab ?agent ?object))
8
9  ;the agent has to reach the location of the object
10 ((agent ?agent) (object ?object)
11  (location ?from)(location ?to)
12  (at ?agent ?from)(at ?object ?to))
13 ;go to the object location and grab it
14 ((go ?agent ?from ?to)(grab ?agent ?object))
15 )
```

When the system is assigned a behavior directive that encompasses the task of taking an object (for example `(take agent1 folder)`), the parameters of the method are bound to the corresponding scene entities (here, `agent1` and `folder`) and the method is refined until a sequence of primitive actions is obtained ("`grab`" or "`go` and `grab`").[5]

The description of the scene is a set of ground formulae, constrained to the template list below, designed to meet the requirements of the staging task (where the director puts the entities on stage according to a certain layout):

- The list of *characters* that appear in the scene;
- The list of *objects* in the scene, such as pieces of furniture, props, etc.

---

scription language: see [28] for the mapping of this format onto the standard format for plan representation, PDDL.

[5]Notice that the planning system does not support explicit entity types in the description of the plans, so it is does not provide a mechanism to enforce the binding to action parameters onto the scene entities. For this reason, in AnimaTricks, we created a simple taxonomy of entity types, that can be used in the specification of plans to guarantee that their parameters are bound to the appropriate scene entities when the plan is applied. In the example, the `?agent` and `?object` parameters are constrained to be, respectively, instances of the `agent` and `object` types.

- The list of relevant scene *locations* (coordinates in the Animation engine).
- The *positions* of characters and objects, i.e., the location in which their are situated.
- The list of *events* to which the character is to react.

Executor. Given the plan generated by the Planner (e.g., `!sit-down agent1 desk2, !grab agent1 pen1, !write-line agent1 sheet1, ...`), the Executor module retrieves from the Action Catalogue the corresponding A2A (Action to Animation) mapping, a procedural definition where the action is mapped onto some animation construct expressed in the AnimaTricks language (see below). A2A mappings, issued from the breakdown process carried out by the animator and the 3D programmer, are mutually exclusive (to avoid conflict resolution strategies, not particularly relevant in a storytelling context). The A2A mapping is interpreted in its sequential/parallel structure to produce a sequence of calls to the animation primitives and generate the final blended animation.

3.3. The AnimaTricks Language: A2A mapping and animations

The main features of the animations are:

- animations operate on skeleton joints, blend-shape weights, and on the global position and rotation of the character in space;
- primitive animations are defined by loading data-driven stored animations or implementing procedures for generating simple animations for global position, rotation and IK end-effectors (e.g., hands reaching specific locations, eye gaze, finger pointing, ...);
- non primitive animations can be defined by: sequencing animations (procedural blending achieves smooth transitions); layering animations, with a control of priority levels for overlapping DOFs; iterating animations. Sequencing and layering form animations too, thus supporting hierarchical structures;
- synchronization over animations is only realized through the specification of sequential / parallel hierarchical structures; the language does not require an explicit encoding of timing; durations are intrinsic in

data-driven animation clips, while for procedural animations speed is specified instead of duration.

A2A definitions map actions onto complex animations plus a set of parameters. Parameters depend on the schematic structure of an action as encoded by the corresponding frame (consulting offline the FrameNet lexical resource [29] or some process ontology [30]): for example the action "walk" can expressed by specifying a path along which the action occurs, the source and destination locations of the action, the area in which the walking action takes place. The work described in [31] exemplifies how characters' actions can be mapped onto ontology concepts via the lexical mediation provided by FrameNet.

The syntax of the AnimaTricks language that encodes such concepts includes the Primitives Types, the Objects, and A2A definitions. The Primitive Types are: *String*, $float$, *int*, *Point*, *Rotation*, *Anim*, $List < T >$, where the $< T >$ template-like syntax denotes that containers are specialized at instantiation according to the contained type.

Objects are named sets of attributes (such as position and rotation),

$Point \leftarrow location\_of(obj\_name : String)$
$Rotation \leftarrow rotation\_of(obj\_name : String)$

that can be grouped in lists:

$List < T > \leftarrow$ "["$(element : T) *$ "]"

Basic animations can be obtained through the *retrieval* of a clip from a repository of animation clips:

$Anim \leftarrow clip\_animation(id : String)$

or through pure procedural animation, e.g., an IK end-effector (or handle) moving or rotating to a specific value:

$Anim \leftarrow move\_handle\_to(handle : String, pos : Point, speed : float)$
$Anim \leftarrow rotate\_handle\_to(handle : String, rot : Rotation, angular\_speed : float)$

An object moves through the space following a *path* of points, at a certain speed (in meters per second). The duration of the animation is computed on the length of the whole path:

$Anim \leftarrow follow\_path(points : List < Point >, speed : float)$

rotate_to and *rotate_towards* create an animation that makes the character rotate to the specified rotation or towards the specified point at the

specified turn speed (in degrees per second), respectively, typically used to align objects:

$Anim \leftarrow rotate\_to(rot : Rotation, turn\_speed : float)$

$Anim \leftarrow rotate\_towards(p : Point, turn\_speed : float)$

Animations can be structured in time through:

1) the repetition of the same animation, using the definition

$Anim \leftarrow repeat(a : Anim, times : int)$, using $times$ -1 for infinite loops.

2) the sequentiality *blending* of several animations, via the function

$seq(anims : List < Anim >)$.

3) the parallel construct, with a first-defined / high-priority fashion, meaning that (similarly to Improv) DOFs controlled by firstly defined animations will have precedence on the same DOFs affected by following animations in the provided list. The syntax is:

$par(anims : List < Anim > [, endsync : String])$

The optional "endsync" parameter defines the duration behavior of the parallel sequence. Allowed values are "all" (default) and "first". Their meaning follows the specification of the SMIL language.[6] The last three functions correspond to the repetition, parallelism, and sequence composition constructs in the MURML language [32].

Actions are procedurally defined as animations, possibly structured according to some sequential or parallel construct. Here are some examples. The action Say-goodbye (below, line 1) is a basic form of non-parametrized invocation of a data-driven animation stored with the name "goodbye" (line 2).

```
1 Say-goodbye() {
2   clip_animation("goodbye")
3 }
```

Another example is the Grab-one-hand action, presenting both parameters (line 1) and sequencing (line 2): the definition of this action requires the hand of the actor to reach the target location (line 3) before the actor closes its hand (line 4).

[6] http://www.w3.org/TR/2008/REC-SMIL3-20081201/smil-timing.html#Timing-endsyncAttribute [last visited on 8 May 2013]

```
1 Grab-one-hand(target:Point) {
2   seq([
3     move_handle_to("hand", target, DEFAULT_HAND_SPEED)
4     clip_animation("hand_closed") ;
5   ])
6 }
```

If the author wishes to modulate the speed of the hand movement (to have a quicker gesture) a float multiplier can be used (e.g., DEFAULT_HAND_SPEED * 1.7)

The Walk action (line 1 below) presents a combination of sequential and parallel movements. It is a sequence (line 2) of a rotation towards the point to reach (line 3) followed by the actual walking animation. The walking animation is built by parallelizing (line 4) a looping walking cycle (line 5), with the primitive animation moving the agent between two specified points (line 6). The "first" directive (line 8) says that the parallel animation terminates as soon as the first of its component ends.

```
 1 Walk(from_location:String, to_location:String) {
 2   seq([
 3     rotate_towards(position_of(to_location), DEFAULT_ROT_SPEED * 3)
 4     par(
 5       [ repeat(clipAnimation("walk_cycle"), -1)
 6         follow_path( [ from_location to_location ], DEFAULT_WALK_SPEED)
 7       ],
 8       "first"
 9   ])
10 }
```

The Animation Engine is provided by the Enthusiasm project[7], an open source platform that supports the authoring of 3D real-time interactive virtual environments. Enthusiam contains a 3D engine with high-level functionalities for building applications based on real-time 3D technologies: real-time 3D rendering (on top of DirectX or OpenGL), import assets from popular 3D authoring tools (such as Maya, 3DStudio, Blender, and more), spatialized audio, physics, multiplatform support (Windows, Linux, MacOS X). Original features are: a simplified scene management, both C++ and Java APIs, effortless integration in Java AWT/Swing interfaces, and the character animation engine described in this paper.

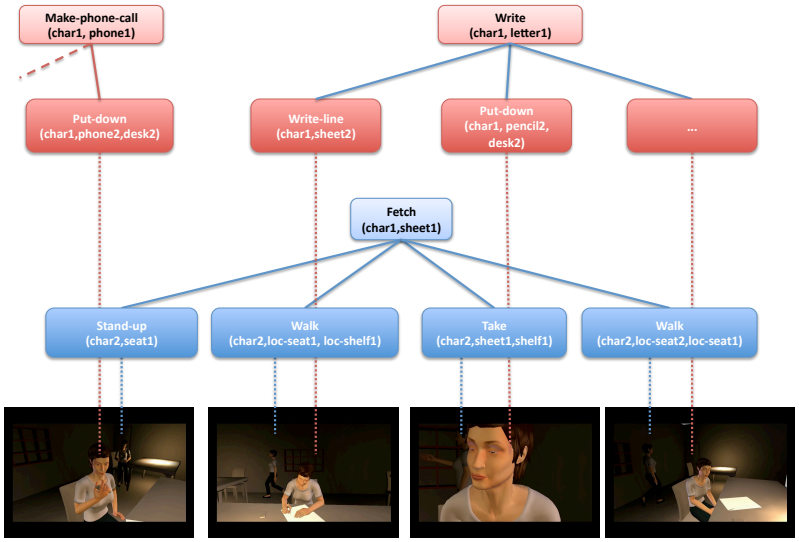

Figure 3: Snapshots of the video automatically generated by the system from the planning goal.

4. Case study: animation of extra characters

We employed our system to conduct an experiment on the creation of extra characters in serial productions (TV series, video games, etc.).[8] The validation test had a duration of one month, involving two 3D artists (modeling and tuning of the character with respect to the production needs), one senior animator (animation and labelling), two software developers (programming the Animation engine), one visual artist (setup, lighting, and shooting of the scenario), an AI engineer. Figure 3 contains an example animation taken from one of the generated sequences of actions. In the snapshot, two characters do office work, answering the phone and writing (char1) or standing up to fetch some paper sheets (char2, in background). The upper part of the figure shows the plan generated by the planner; in each shot, the characters

[7]http://enthusiasm.sourceforge.net/ [last visited on 8 May 2013]

[8]The software and data employed in the experiment can be downloaded at http://www.cirma.unito.it/animatricks/downloads/project/

are connected by the dashed lines to the plan actions they are executing.

The experiment is tailored on a real production scenario. Since we assume that the reuse of animated behaviors across episodes/environments/sessions, is relevant for evaluating the impact of the system, we included re-use aspects in the experiment design. So, as a preliminary phase, we asked an author to write down a script by inserting actions that typically recur in the series productions (for example, sitting at the desk and writing some letters, etc). Given this script, a 3D character was designed, modeled and rigged according to the requirements issued by the animator, and animated by using traditional techniques. Then, we proceeded through three main phases:

1. creation of the repository of animations, obtained by breaking down the actions, producing the basic animation clips or poses, and labeling the animation for subsequent reference;
2. testing the offline pipeline (see Section 3.1): first, behavior definition, through the authoring of a set of complex, goal directed character behaviors; then, catalogue creation, through the mapping of the behavior primitives onto the animation repository;
3. testing the online pipeline: staging and animation of the character on a set of authorial directives, conducted by using the AnimaTricks system (see Section 3.2).

Each phase involved a specific set of challenges. **Phase 1** (segmentation of existing animation into action units and labeling) proved that identifying meaningful actional units for reuse in an animation clip is a reasonable task. Given an animation composed of 40 seconds (1000 frames), the animator identified 43 actions (e.g., writing, grabbing, etc.) and 27 poses (e.g., surprise takes, eye blinks, etc.), most of which involved only the facial expressions.

Phase 2 of the experiment faced the challenge of the authoring of behaviors by the AI engineer under the authorial control of the animation director. The behavior described by the writer was encoded in a plan library, then tested onto different goals and scene configurations; the refinement went on until the animation director judged the result satisfactory for the coherence and recognizability of the generated action sequences. The plan library contained 17 complex actions (methods) and 21 primitive actions (operators), that were mapped onto the actions in the catalogue. The planner was tested on 20 different scenarios, where the AnimaTricks agents were given different tasks (or the same tasks with different parameters). Output plans ranged

from 16 to 32 actions; 5 scenarios were selected to run the evaluation based on the procedural animation techniques required. The plan library included actions such as entering, sitting at the desk to accomplish several tasks, like doing or receiving phone calls, hand-writing letters and notes, getting up to take objects (pen, sheets, etc.) when necessary.

For example, a fragment of the plan represented in Figure 3 generated the following sequence of operators:

```
[ 11 ]  (!Talk agent1)
[ 12 ]  (!Say-goodbye agent1)
[ 13 ]  (!Put-down agent1 phone1 desk1)
[ 14 ]  (!Grab-one-hand agent1 pen1)
[ 15 ]  (!Write-line agent1 section1 sheet1)
[ 16 ]  (!Write-line agent1 section1 sheet1)
[ 17 ]  (!Finish-page agent1 section1 sheet1)
[ 18 ]  (!Put-down agent1 pen1 desk1)
[ 19 ]  (!Stand-up agent1)
[ 20 ]  (!Walk agent1 desk-loc shelf-loc)
```

The creation of the Action Catalogue involved the programming of A2A rules for the actual reuse of the stored animation data. The animator and the 3D programmer analyzed the structure of each primitive action included in the plan library to translate it into the animation language, possibly reusing clips stored in the Animation Data Repository. Most of the actions had to be produced from scratch as animation clips. This is the case for primitive actions, such as `!Talk`, `!Put-down`, `!Write-line`, corresponding to the units *Uttering*, *Taking*, *Writing*, identified during the breakdown of the original animation. For other actions, however, an existing clip (or pose) was embedded in an A2A mapping rule, such as *walk* and *grab-one-hand*. In the experiment, we developed 9 direct correspondences between action and animation and 3 A2A rules, with a use of action parameterization. However, given its short duration, the animated scene did not include any reuse of animations across different actions.

Phase 3 directly challenged the functioning of the animation system (integration of components, computational resources, etc.). First, the main character has been animated by composing the animations retrieved from the repository (instead of performing full length manual animation). Then, given two background characters, the system was assigned a complex directive for each character; each directive yielded 8 different plans, and one plan was chosen for each character by the director. We were able to optimize two cases:

- The main character was animated on main gestures and poses and blended subsequently. We were able to automatize 20 out of the 40 seconds of animation, thus halving the animation work (though some interpolation is already in use in major 3D authoring software). This means saving about 2-day work of the senior animator.

- We also provided 30 seconds of animation for each of the two secondary characters in the scene. Animation time for secondary characters is normally lower that for primary characters, so this correspond to roughly 3-day work of a junior animator.

These times must be compared to the time needed by the senior animator to segment and label the script (1 day), the time needed by AI engineer to encode and debug the plans describing the character behavior (2 days), and the time needed to develop A2A rules for the Action Catalogue (1/2 day). This translates in having converted 5 days of animation work into 3 and 1/2 days of multiple specialized work. Given the short duration of the movie (a more thorough experiment would require a very expensive budget), these results can be considered a success. We believe that longer durations of serial animations, also across several episodes involving the same characters, would produce greater time savings.

At the end of the experiment, in order to evaluate the quality of the obtained animation, we asked a panel of animation experts and producers to assess if the quality was acceptable for the creation of extra characters. The resulting scene, together with the production process, was presented to a focus group of five animation producers, videomakers, and trainers, in a public panel. The aim of the focus group was to assess the quality of the animation produced through the reuse of action animations, the feasibility of the pipeline (including the semantic labeling and the search and reuse phase), and to validate the production scenarios for which the system was designed (offline animation and animation of extra characters). The experts evaluated its quality and potential for use in the animation industry and the impacts of the technique for the animation languages. According to the comments we collected, the system has a high potential in the animation production, for both main and extra characters, in the case of TV series productions, especially if coupled with animation data extracted from motion capture takes. However, it should be noticed that the major problems concern the creation of the animation catalogue and the classification system that allows

for a fast retrieval of the animation segments. Past experiences in series production that tried to create a repository of animation segments lead to a waste of time in classification and retrieval. From the discussion, it seems that an action–based storing and retrieval of animations would facilitate the task of animators: in most past experiences, producers reported about the frustration of animators in retrieving their own work, with the consequence that most of them tended to animate the same action again.

5. Conclusions and Future Work

In this paper we described a system for the modularization and automation of character animation starting from a high–level specification of their behavior. The system is based on a reference architecture that integrates an AI decision making component to generate the agent's behavior and an animation component that transforms the agent's behavior into perceivable acts in a 3D graphic environment.

Together with the system architecture, we presented a production pipeline that involves traditional figures, such as 3D artists and animators, to work with AI engineer and 3D programmers. The pipeline requires authors, AI engineer, animators and 3D programmers to cooperate to the representation of the agent's behavior both at the decision-making and at the animation level. The work of the professional roles in the pipeline is accompanied by declarative languages that support each production phase. For character animation we presented a declarative animation language, with a set of rules that map the agent's actions, generated by the AI component, to hand-made and procedurally generated animations.

We also described an experiment in which the pipeline and the system were used for the animation of secondary characters in serial productions. This case study provided a test bed for the evaluation of the system and gave encouraging results, both on the feasibility of the pipeline and time saving.

The current system has some limitations. First of all, the rules that map the actions in the catalogue to the animation language must be coded by hand. In order to alleviate this task, we will investigate the possibility of a semi-automatic translation given the structure of the action itself. The mapping of parameters from plan actions to animations, addressed by [33], also deserves further investigation. Second, the semantic labels attached to actions are not exploited in the current system: so, in the future, we are

planning to use state-of-the-art technologies to support semantic access to the Action Catalogue. Third, currently the system does not support multi-agent coordinated actions. All the animated behaviors must be monitored by the animation director and generated again in case of conflicts in general and any collisions in particular. Finally, the system does not support interactivity, since the resulting plan devised by the decision making component is directly executed in the 3D virtual world with no reconsideration in case of failures. Since the paradigm of HTN planning can easily be adapted to deal with re–planning [34, 35], we intend to expand the system to interactive animated agents.

References

[1] L. Manovich, The Language of New Media, The MIT Press, 2001.

[2] I. Kerlow, The Art of 3D Computer Animation and Effects, Wiley Publishing, 4th edition, 2009.

[3] R. Parent, Computer Animation: Algorithms and Techniques., Morgan Kaufmann, 2007.

[4] Y. Abe, M. da Silva, J. Popović, Multiobjective control with frictional contacts, in: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation, SCA '07, Eurographics Association, Aire-la-Ville, Switzerland, 2007, pp. 249–258.

[5] W. Swartout, D. Traum, R. Artstein, D. Noren, P. Debevec, K. Bronnenkant, J. Williams, A. Leuski, S. Narayanan, D. Piepol, et al., Ada and grace: toward realistic and engaging virtual museum guides, in: Intelligent Virtual Agents, Springer, pp. 286–300.

[6] S. Kopp, L. Gesellensetter, N. Kraemer, I. Wachsmuth, A conversational agent as museum guide - design and evaluation of a real-world application, in: 5th International Working Conference on Intelligent Virtual Agents (IVA'05).

[7] A. Bogdanovych, J. Rodriguez-Aguilar, S. Simoff, A. Cohen, Authentic interactive reenactment of cultural heritage with 3d virtual worlds and artificial intelligence, Applied Artificial Intelligence 24 (2010) 617–647.

[8] R. Aylett, M. Vala, P. Sequeira, A. Paiva, Fearnot!–an emergent narrative approach to virtual dramas for anti-bullying education, LNCS 4871 (2007) 202.

[9] L. Ieronutti, L. Chittaro, Employing virtual humans for education and training in X3D/VRML worlds, Computers & Education 49 (2007) 93–109.

[10] M. Mateas, A. Stern, Integrating plot, character and natural language processing in the interactive drama Façade, in: TIDSE 03.

[11] D. Pizzi, F. Charles, J. Lugrin, M. Cavazza, Interactive storytelling with literary feelings, in: The second International Conference on Affective Computing and Intelligent Interaction (ACII2007), Springer, Lisbon, Portugal, September, 2007.

[12] V. Lombardo, F. Nunnari, R. Damiano, A Virtual Interpreter for the Italian Sign, in: Intelligent Virtual Agents: 10th International Conference, IVA 2010, Philadelphia, PA, USA. Proceedings, Springer, p. 201.

[13] R. Niewiadomski, E. Bevacqua, M. Mancini, C. Pelachaud, Greta: an interactive expressive eca system, in: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2, International Foundation for Autonomous Agents and Multiagent Systems, pp. 1399–1400.

[14] J. Skorupski, Storyboard authoring of plan-based interactive dramas, in: Proceedings of the 4th International Conference on Foundations of Digital Games, FDG '09, ACM, New York, NY, USA, 2009, pp. 349–351.

[15] M. Mateas, A. Stern, Writing façade: A case study in procedural authorship, Second Person: Role-Playing and Story in Games and Playable Media (2004) 183–208.

[16] M. Cavazza, F. Charles, S. Mead, Interacting with virtual characters in interactive storytelling, in: Proc. of the First Int. Joint Conf. on Autonomous Agents and Multiagent Systems.

[17] S. Kopp, L. Gesellensetter, N. C. Krämer, I. Wachsmuth, A conversational agent as museum guide - design and evaluation of a real-world application., in: Intellingent Virtual Agents, pp. 329–343.

[18] F. Dignum, J. Westra, W. van Doesburg, M. Harbers, Games and Agents: Designing Intelligent Gameplay, International Journal of Computer Games Technology 2009 (2009).

[19] K. Perlin, A. Goldberg, Improv: a system for scripting interactive actors in virtual worlds, in: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, volume SIGGRAPH '96, ACM, ACM press, New York, NY, USA, 1996, pp. 205–216.

[20] D. Isla, R. Burke, M. Downie, B. Blumberg, A layered brain architecture for synthetic creatures, in: International Joint Conference on Artificial Intelligence, volume 17, Citeseer, pp. 1051–1058.

[21] A. Loyall, W. Reilly, J. Bates, P. Weyhrauch, System for authoring highly interactive, personality-rich interactive characters, in: Proc. of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation, pp. 59–68.

[22] N. I. Badler, R. Bindiganavale, J. Allbeck, W. Schuler, L. Zhao, M. Palmer, Parametrized action representation for virtual human agents, in: J. Cassell, J. Sullivan, S. Prevost, E. Churchill (Eds.), Embodied Conversational Agents, The MIT Press, Cambridge, Massachussets, 2000, pp. 256–284.

[23] S. Kopp, B. Krenn, S. Marsella, A. Marshall, C. Pelachaud, H. Pirker, K. Thórisson, H. Vilhjálmsson, Towards a common framework for multimodal generation: The behavior markup language, Lecture Notes in Computer Science 4133 (2006) 205.

[24] J. Cassell, H. Vilhjálmsson, T. Bickmore, BEAT: the behavior expression animation toolkit, in: Proceedings of the 28th annual conference on Computer graphics and interactive techniques, ACM New York, NY, USA, pp. 477–486.

[25] A. Heloir, M. Kipp, Embr – a realtime animation engine for interactive embodied agents, in: LNCS, Intelligent Virtual Agents, Springer, 2009.

[26] D. Nau, T. Au, O. Ilghami, U. Kuter, J. Murdock, D. Wu, F. Yaman, SHOP2: An HTN planning system, Journal of Artificial Intelligence Research 20 (2003) 379–404.

[27] D. Nau, T.-C. Au, O. Ilghami, U. Kuter, H. Munoz-Avila, J. W. Murdock, D. Wu, F. Yaman, Applications of shop and shop2, IEEE Intelligent Systems 20 (2005) 34–41.

[28] R. Alford, U. Kuter, D. Nau, Translating htns to pddl: A small amount of domain knowledge can go a long way, IJCAI, July (2009).

[29] C. Baker, C. Fillmore, J. Lowe, The berkeley framenet project, in: Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics-Volume 1, Association for Computational Linguistics, pp. 86–90.

[30] I. Niles, A. Pease, Mapping WordNet to the SUMO ontology, in: Proceedings of the IEEE International Knowledge Engineering conference, pp. 23–26.

[31] M. Cataldi, R. Damiano, V. Lombardo, A. Pizzo, Lexical mediation for ontology-based annotation of multimedia, in: New Trends of Research in Ontologies and Lexical Resources, Springer, 2013, pp. 113–134.

[32] A. Kranstedt, S. Kopp, I. Wachsmuth, Murml: A multimodal utterance representation markup language for conversational agents, in: Proc. of the AAMAS Workshop on Embodied conversational agents–Lets specify and evaluate them.

[33] S. P. Cash, R. M. Young, Bowyer: a planning tool for bridging the gap between declarative and procedural domains., in: Proc. of AIIDE 05.

[34] G. Boella, R. Damiano, A replanning algorithm for decision theoretic hierarchical planning: principles and empirical evaluation, Applied Artificial Intelligence 22 (2008) 937–963.

[35] R. Van Der Krogt, M. De Weerdt, Plan repair as an extension of planning, in: Proc. of the Int. Conf. on Automated Planning and Scheduling.