Action Description for Animation of Virtual Characters

Arnd Vitzthum, Heni Ben Amor, Guido Heumer, Bernhard Jung

Virtual Reality and Multimedia Group
Institute for Informatics, TU Bergakademie Freiberg

Abstract: In this paper we present XSAMPL3D, a novel language for the high-level representation of actions performed on objects by (virtual) humans. Actions are encoded in a compact and human-readable XML-format which is a suitable basis for the synthesis of virtual human animations. XSAMPL3D descriptions can be generated automatically from captured VR interaction data or created manually for rapid prototyping of animations.

Keywords: Virtual Humans, Motion Synthesis, Action Representation

1 Introduction

Previous work on the animation of virtual humans performing object manipulations has led to two main approaches: data-driven methods, using motion-capture data, and model-driven methods, synthesizing animations at runtime. Usually, data-driven approaches suffer from limited adaptability to new situations, such as changing environments and different character proportions, e.g. the well-known retargetting problem [Gle98]. Model-driven approaches on the other hand allow for high adaptability but often result in unnatural, 'robotic'-looking animations. In earlier work, we described a Virtual Reality-based approach to character animation that aims to combine the strengths of the data- and model-driven approaches: First, a VR user demonstrates the manipulations of scene objects. The user's movements *and*, complementing traditional motion capture, interactions with the objects of a virtual environment are recorded via VR tracking devices, including data gloves or equivalent finger tracking systems (in contrast to classical VR manipulation techniques, where the selection of an object has to be confirmed explicitly, we focus on natural interaction by using grasp recognition). Then, the recorded motion and interaction data are abstracted to an intermediate *action representation*. Finally, animations of virtual characters are generated from these action representations through behavioral animation techniques. Following research results from the field of imitation learning, where a distinction is made between imitation on the level of mere movements as opposed to imitation of actions on objects, action = movement + goal [Arb02], we call this approach *action capture* [JAHW06].

As this paper's main contribution, a detailed account of the action representation language XSAMPL3D is presented. As the main focus of action capture lies on the recording and animation of realistic manipulations of objects, XSAMPL3D representations center on action/object pairs, supplemented by information allowing for smooth animation of virtual

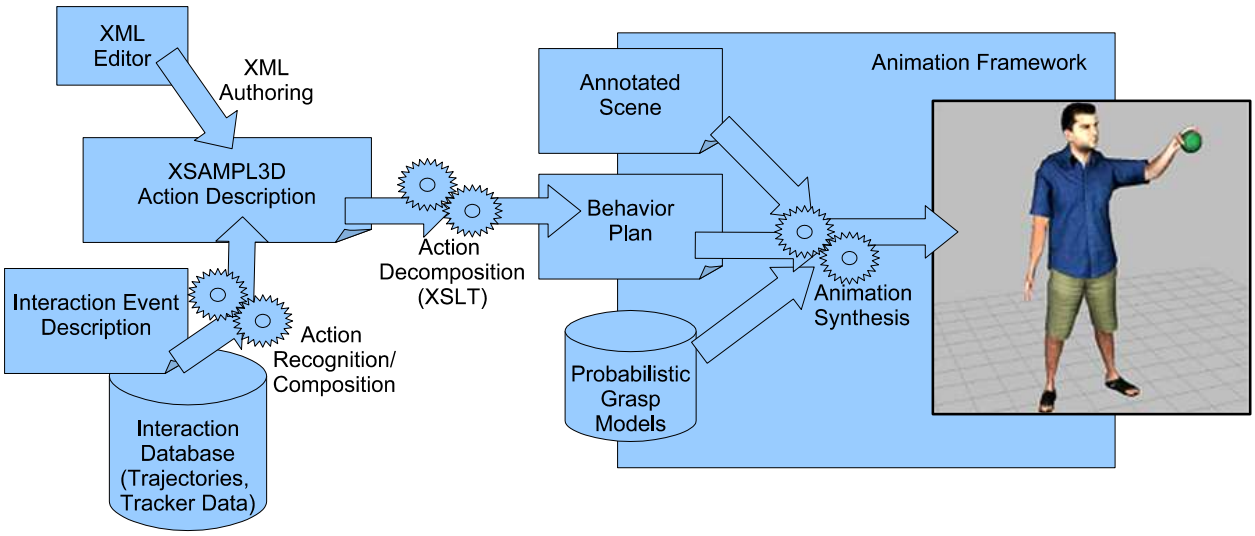

Figure 1: Action Capture - action representation and animation synthesis.

characters such as timing and trajectory data. The automatic derivation of XSAMPL3D descriptions from user interactions in VR (lower left corner in Figure 1) builds on the detection of basic interactions such as *grasp*, *move*, and *release* which are then combined to complete actions on the involved object. XSAMPL3D representations are however also compact enough to be directly authored by an animation designer (upper left corner in Figure 1), e.g. for purposes rapid prototyping of animations showing virtual humans performing object manipulations.

2 VR Infrastructure

Even though the action language presented here can also be used to rapid-prototype animations by manually creating actions description files, it is normally embedded in the action capture architecture. This architecture poses several demands on the infrastructure of the virtual reality system it runs on.

2.1 Scene Representation: Annotated Objects

An important component of an action description and animation system is formed by the objects on which actions are performed. In the sense of Arbib's formula $action = movement + goal$ [Arb02], objects are the goals of the actions. In computer graphics objects are mostly described as geometric models with additional information for rendering like materials, texture and other maps, etc. However, for the kind of complexity involved in the action capture method this is not enough. A description method is called for that allows for integration of other aspects about an object that go beyond mere graphical appearance, like physical properties, joint information for articulated objects and semantic annotations to support the grasping algorithms. To provide a convenient mechanism for declaration of all these different aspects of higher-level information about scene objects, the concept of

annotated objects has been introduced [WHAJ06]; for an extension to articulated objects such as control actuators like knobs, switches etc. see [Gom08]. Object type descriptions are written in an XML-based representation structure and stored in a common database for reference.

2.2 Interaction Recording

Since our approach for animation synthesis is primarily data-driven there needs to be some way to record VR interaction data. Most importantly *arm trajectory* and *hand posture* data during grasping are recorded. In our system all interaction data is stored persistently in an interaction database organized by recording sessions. Recording sessions are subdivided into channels each of which represents a certain single stream or element of the interaction data. This could be an input device like an optical tracker or a data glove, or motion data of a certain part of the body like hand postures or hand trajectories. The latter are recorded in a hand trajectory channel of which there is one for each hand. Trajectories within the channel are segmented by pauses in the motion or by object contact.

Hand posture data can be annotated with the grasp type used. In this way it can be used to train a grasp recognition classifier or as example data to create a low-dimensional posture model for a particular grasp type (see section 4). All grasp types are specified with respect to a certain grasp taxonomy. During the course of the virtual workers project several taxonomies have been explored. Currently the Schlesinger taxonomy [Sch19] is used for its simplicity and since it produces reliable classification results.

The result of the intermediate interaction analysis process is the detection of basic interactions (like reach-motions, grasping, releasing and manipulations of objects) which form the 'building blocks' of complete action descriptions. The details of the interaction analysis process go beyond the scope of this paper and the process is only mentioned here for the sake of completeness.

3 The XSAMPL3D Action Language

In order to realize a simple exchange and a compact representation of action descriptions, we defined an XML-based language called XSAMPL3D (XML Synchronized Action MarkuP Language for 3D). XSAMPL3D offers a human-readable syntax at a high level of abstraction. On the one hand, this facilitates the authoring of action descriptions and the post-editing of previously captured action sequences. On the other hand, if XSAMPL3D actions are derived from lower level data such as interaction events, trajectories and motion records, some of the details contained in the original data are lost. To compensate for this drawback, a multi-layer architecture was realized within the action capture framework, which allows for referencing lower-level information from higher-level layers. For example, an action description may contain links to the underlying interaction event layer. Using this mechanism, an animation player is able to synthesize an action animation at different levels of detail: From a 'pure'

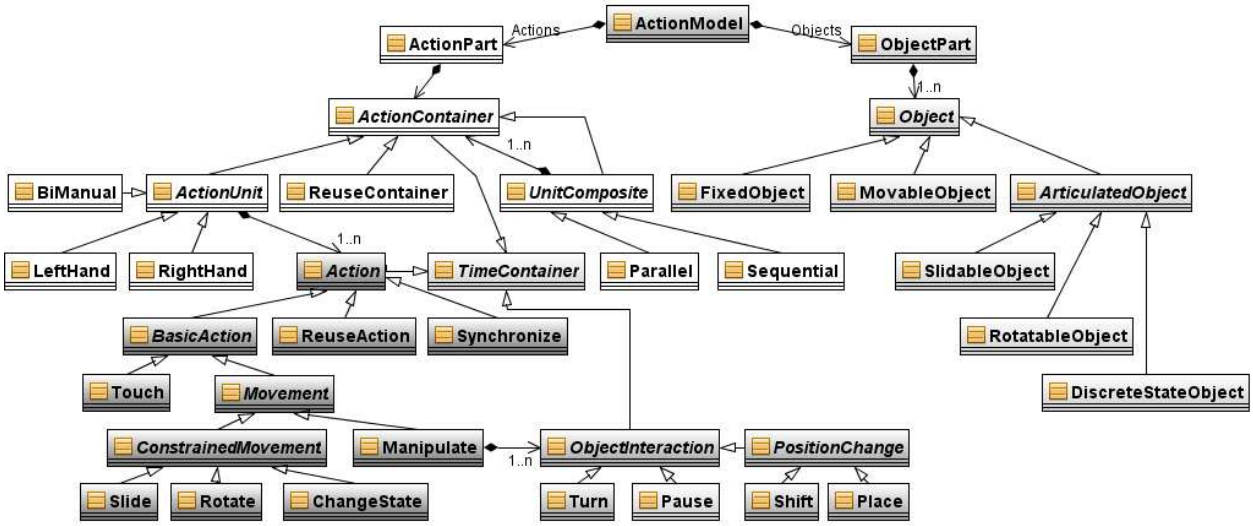

Figure 2: Simplified overview of the XSAMPL3D schema elements.

XSAMPL3D description, an animation can be generated which resembles roughly a recorded human motion (low level of detail). If the XSAMPL3D document contains additional links to interaction event data it is possible to reproduce the original motion very precisely (high level of detail). However, in either case an XSAMPL3D document must provide sufficient information to generate a natural looking animation.

Until now, XSAMPL3D comprises methods for the specification of one- or two-handed object manipulations. Actions can involve the displacement of objects. Currently, we do not consider indirect manipulation of objects (i. e. manipulating an object by using another object, such as displacing an object with a tool).

XSAMPL3D was specified as an XML schema. The schema defines a hierarchy of action types and additional elements related to the composition and synchronization of actions. It can be seen as a framework which delivers the components for building complex action scenarios. Figure 2 presents a simplified overview of the different action types.

XSAMPL3D was designed with extensibility in mind. Since very specific scenarios require very specific actions, it is almost impossible to specify a complete set of all imaginable actions. For example, while in a sports scenario an action *throw object* could be useful, in a car driving scenario such an action makes little sense. Thus, only a small set of actions was defined which allows us to describe the use cases we primarily address in our project, such as ergonomic studies on virtual machine prototypes. At schema level, a developer can extend XSAMPL3D by deriving new action types from the predefined ones in order to adapt the language to new situations. In addition, at instance level new actions can be composed using a set of interaction elements as described later in this section.

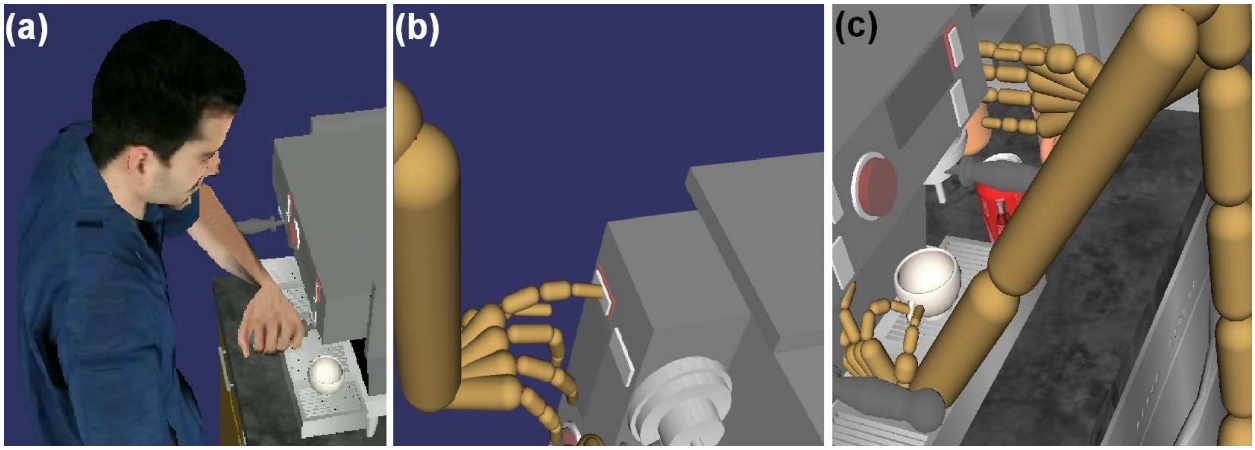

Figure 3: Espresso machine example. (a) The virtual human attaches a porta filter to an espresso machine. (b) The virtual human (skeletal representation) presses a button to fill a cup. (c) The cup is taken out of the machine.

3.1 Example

As an example to demonstrate the application of XSAMPL3D we have chosen a brief scenario (Figure 3). We developed this example among others in a practical course together with students at the TU Freiberg in order to show the feasibility of our approach.

3.2 Action Specification

The specification of the action format includes different aspects, such as (manipulated) objects, different action types, action composition, synchronization and timing.

Objects can be divided into several classes: *articulated objects* (e.g. control actuators), *fixed objects* and *movable objects*. Articulated objects have specific movement constraints which are defined in a separate *annotated object document* (see section 2). For example, a slider can only be moved along one axis and has a minimum and a maximum position. Fixed objects cannot be moved at all while movable objects can be moved arbitrarily (e. g. a cup).

An *action* describes the interaction of the (virtual) human's hand with a particular object. Conceptually, an action consists of different phases: *Reaching* (approaching the object), *grasping*, *object interaction* and *releasing* the object. Different action types can be distinguished: *constrained object movements* (`Slide`, `Rotate`, `ChangeState`), *unconstrained object movements* (`Manipulate`) and actions which don't result in an object displacement (`Touch`). Some action types can be only performed on special objects. For instance, constrained movements refer to the corresponding articulated object types. Unconstrained movements normally result in a change of the position and/or orientation of a movable object.

We focus on the manipulation of freely movable objects here since a more detailed discussion of articulated objects and the corresponding actions would go beyond the scope of this paper.

The manipulation of an object can involve different interactions, such as `Place`, `Shift` or

`Turn`. The additional *Pause* element enables the insertion of a pause of a certain duration into a sequence of interactions. `Place` places an object at a certain position expressed in world coordinates (default), its own local object coordinates or relative to another object. It is also possible to specify the final orientation of the manipulated object. `Shift` is comparable to `Place`, with the difference that the object is moved over a planar surface (such as a computer mouse moved over a desk). A `Turn` interaction results in an orientation change of the object by specifying a turn angle and an optional turn axis. The turn axis is the "Up"-axis of the object by default. For example, in the kitchen scenario, the porta filter is first moved to the espresso machine and attached to it afterwards by a turn. The corresponding action description looks as follows:

```
<Manipulate ID='AttachFilter' graspType='schlesinger:cylindrical'
            object='PortaFilter' reachDuration='0.5'>
    <Place  duration='1.0' orientation='0 0 1 0.785'
            position='0 0.1 0.1' posRelativeTo='EspressoMachine'/>
    <Turn   duration='0.5' angle='-0.785'/>
</Manipulate>
```

An action has two timing related attributes: `reachDuration` and `duration` (part of the contained interactions). `reachDuration` represents the time required to position the hand on the target object (reaching phase). During the reaching phase the hand is also preshaped to perform a grasp. The reaching phase has finished if a stable grasp has been established. The grasp type can be defined explicitly by using the action property `graspType` (syntax <grasp-taxonomy>:<grasp-type>). If no grasp type was specified, the animation player has to decide which grasp type can be applied in order to generate a plausible animation. The `duration` attribute of an interaction defines the length of time required to perform the interaction. In the example above it will take 0.5 seconds to turn the porta filter. The sum of all interaction durations and the reach duration represents the overall action duration.

Actions can be grouped together using an *action unit*. An action unit contains a sequence of actions which are executed with the same hand or with two hands cooperatively. In order to enable an appropriate description of action units, three different kinds of action units were defined: `RightHand`, `LeftHand` and `BiManual`. The code example below specifies that all actions of the kitchen scenario are to be performed consecutively with the left hand (simplified).

```
<LeftHand startTime='0' relaxDuration='2'>
  <Manipulate ... object='PortaFilter'>...</Manipulate>
  <TouchObject ... object='Button'/>
  <Manipulate ... object='Cup'><Place .../></ManipulateObject>
</LeftHand>
```

Properties related to timing of an action unit are `startTime` and `relaxDuration`. The `startTime` is the point of time when the first action of the unit starts after the unit was entered. All actions of the action unit are then executed consecutively in the order defined

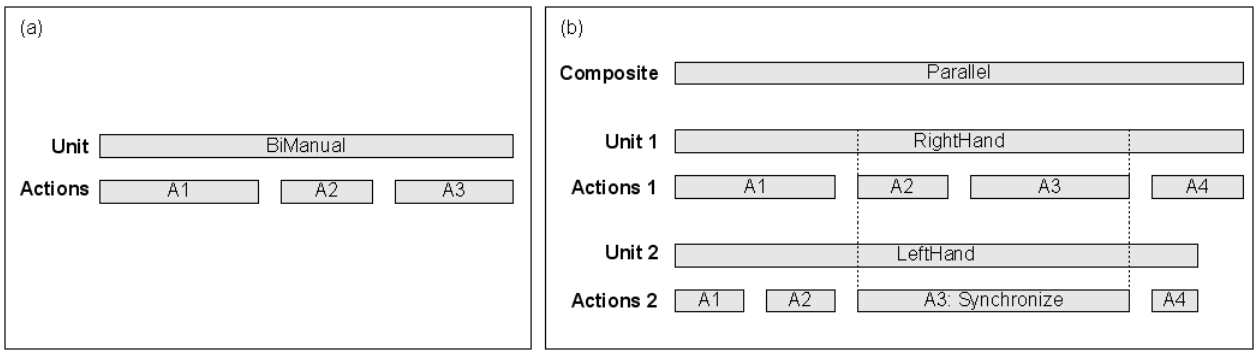

Figure 4: (a) A `BiManual` unit describes a sequence of actions which are performed bimanually. (b) A `Synchronize` action can be used to "synchronize" the movements of one hand with the movements of the other hand to perform selected actions of parallel sequences with both hands. In the example, actions 2 and 3 of unit 1 are performed bimanually.

in the corresponding XSAMPL3D instance document. When the last action has been completed, the action unit enters the *relaxation phase*. The *relaxation duration* of an action unit therefore describes the time required to return to the hand's relaxation position.

Action units themselves can be combined via so called *unit composites*. There are *parallel composites*, in which all actions are executed in parallel, and *sequential composites*, in which all actions are executed successively. The processing of a composite ends if the last (in sequential composites) or longest action unit (in parallel composites) is completed. For example, it would be possible to specify a parallel composite holding a unit which describes the action of the left hand (such as taking the cup out of the espresso machine) and a unit for the right hand, which at the same time is used to perform other tasks. Beside action units, unit composites can contain other composites. For instance, one can define sequential unit composites which are contained in a parallel composite. To realize such a containment hierarchy, the composite design pattern was applied.

`ObjectInteraction` elements, actions, action units and unit composites are so called *time containers*. A time container has its own internal relative time line. Any time container, which is equipped with an ID, can become a kind of a reusable prototype. Instead of inserting a copy of the container description at an appropriate place in an XSAMPL3D instance document, a `ReuseContainer`, `ReuseAction` or `ReuseObjectInteraction` tag with a prototype reference can be applied. An example would be a `ReuseAction` element which references the *AttachFilter*-action defined above.

A very special action type is the `Synchronize` type. `Synchronize` primarily enables the description of bimanual object manipulations. In contrast to the *bimanual unit* (see above in this section), `Synchronize` also allows for partially synchronizing parallel units (Figure 4). Like any other action, `Synchronize` can be inserted into an action unit. A synchronize action contains the ID of the unit to synchronize with and an attribute to specify the duration of the synchronization.

3.3 Action-Behavior-Mapping

Since XSAMPL3D descriptions are independent of a specific animation framework, mappings from actions to behaviors and animations can be realized in several different ways. In our implementation an XSAMPL3D instance document is automatically transformed into an XML behavior plan, which is a suitable format for virtual human animation (see next section). Since XSLT is the de-facto standard for transforming XML documents we use the schema-aware Saxon engine (www.saxonica.com), which allows us to access type information from the XSAMPL3D schema in XSLT. If necessary, the generated XML-plans can be manually refined.

4 Behavior Language and Execution

In our architecture actions describe the animation at a high-level of abstraction. Many informations about *how* the action is executed in detail are not included at this level of abstraction. However, for visualization and replay, such information is indispensable. The transformation of actions into animations is performed using a *behavior-based* animation framework (right-hand side of Figure 1). The framework uses a set of goal-directed behaviors to compute the joint rotation for animating the virtual human. An important property of the behaviors is their context awareness. Depending on the current situation, e.g. positions and orientations of objects, different control parameters for animating the virtual human are generated. During animation synthesis, machine learning and optimization techniques are used in order to guarantee that the generated motions fit the situation and environment at hand, more details are given in Section 4.1. The approach builds on current robot control architectures.

The behaviors contained in the framework include among others the *relax*, *push*, *pick*, *place* and *turn* behavior. The *relax* behavior makes the virtual human take on a relaxed body posture. This is useful for chaining several actions sequences. The *push* behavior can be used to push any objects or buttons. The *pick* behavior makes the virtual human perform a grasping motion in order to pick a given object. Conversely, the *place* behavior is used to move the grasped object to a new position. The *turn* behavior allows the virtual human to turn the grasped object around a given axis.

The framework automatically executes a sequence of behaviors provided in an XML representation. Figure 5 shows the behavior sequence for the kitchen scenario. Each behavior supported by the framework can be instantiated and parameterized in XML. The following XML code shows the instantiation of a Pick behavior:

```
<behavior>
<type>Pick</type>
 <param name="start-time">3</param>
 <param name="end-time">4</param>
 <param name="side">left</param>
 <param name="object">Cup</param>
```

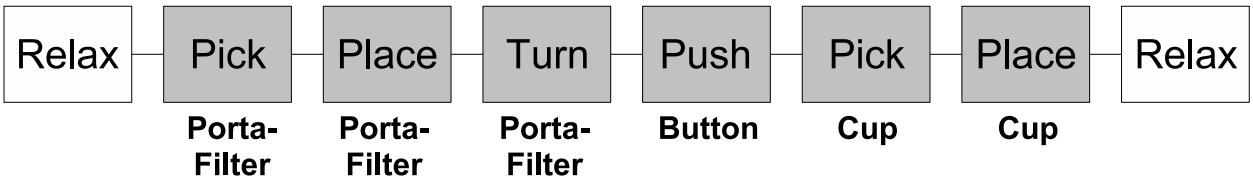

Figure 5: A sequence of behaviors used to create an animation in which a virtual human brews a cup of coffee.

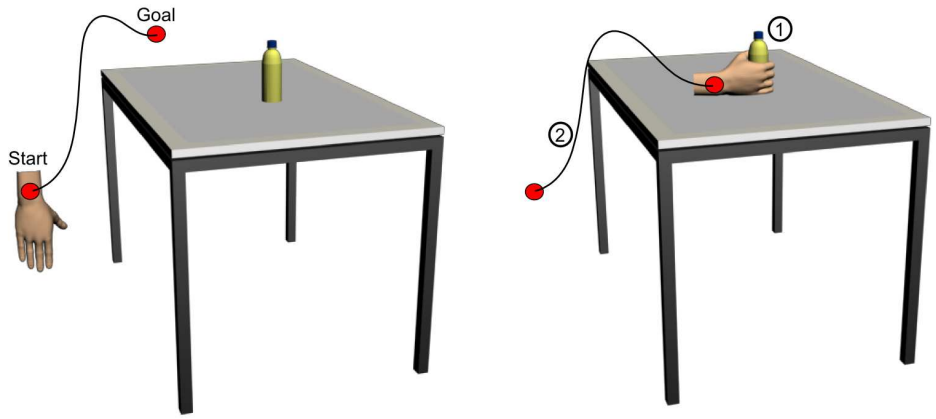

Figure 6: Left: The starting point of the algorithm. The goal is to grasp the object by moving along a recorded trajectory. Right: First a suitable grasp is found using optimization (1). Then, the new position of the wrist is used to retarget the trajectory (2).

```
 <param name="grasp-type">Schlesinger::tip</param>
 <param name="grasp-pre-shape">-1 0 0</param>
 <param name="follow-trajectory-file">trajectory_1_global_left.trj</param>
 <param name="follow-ik-method">heuristic</param>
 <param name="pick-hand-coords-rotation">-1 0 0</param>
</behavior>
```

Apart from the start and end time of the behavior, the user can also supply the name to be picked up and the particular grasp type in which this action should be performed. Typically before grasping, the human hand configuration takes on a preshape followed by the actual closing of the hand. This can be specified by the grasp-pre-shape parameter. The given hand shape is specified as a coordinate of a point in a probabilistic low-dimensional grasp model; for further details on this please refer to [BAHJV08]. The user can further specify which inverse kinematics algorithm should be used. In the following we will roughly explain the grasping algorithm for realizing such a pick behavior. In particular we will show how the algorithm ensures that even new objects or displaced objects can be correctly grasped.

4.1 Grasping Strategy

The synthesis of a new grasping animation consists of two steps, as can be seen in Figure 6. First, a stable and natural looking grasp is generated. This is done using an imitation learning based approach. For this, the user provides some example hand configurations using a data-glove. These recorded demonstrations can be used to synthesize a large number of variations of the intended grasps. Using an optimization algorithm, such as a genetic algorithm, an appropriate grasp can be generated for a given object. In the second step, the new wrist position is used to generate a trajectory leading from the start position of the hand to the new wrist position. Using the algorithm described in [BADV+08], we retarget a previously recorded trajectory to the new wrist position. As a result we have both a trajectory specifying the motion of the wrist during the animation as well as a hand configuration for grasping the object. This is sufficient for generating the animation using well-known algorithms, such as inverse kinematics. Motor programs for different skeletal architechtures, such as the H-Anim format, allow us to replay the animation with a number of available virtual humans.

5 Related Work

The design of XSAMPL3D was influenced by different research domains and standards. In the field of Embodied Conversational Agents (ECAs) researchers separate concepts to describe the agent's intentions ("higher-level" goals) from concepts to describe behaviors. From this viewpoint, XSAMPL3D rather contains behavioral elements (actions) which represent the steps required to reach a certain higher-level goal (such as preparing a cup of espresso), while behavior plans as described in section 4 concretize how (i.e. with which body movements) exactly such a goal can be achieved. Examples of behavior-related XML-formats in the field of ECAs are e.g. MURML [KKW02] or the Behavior Markup Language [VCC+07]. Although these languages focus more on gestures, facial expression, gaze and speech than on the interaction with virtual objects, they share some common aspects with XSAMPL3D, such as the synchronization and timing of activities. Another influence of gesture-related research in XSAMPL3D is represented by the concept of action units. This concept was inspired by Kendon [Ken04], who analogously uses the term *gesture unit* to denote a sequence of connected gestures.

The CONFUCIUS system introduced in Ma and Mc Kevitt [MMK06] can be seen as an example of a character animation framework which is partially based on action descriptions. The authors propose a comprehensive list of hand movements. Some of these movements, especially those involving the manipulation of objects, are comparable to XSAMPL3D interactions. In contrast to XSAMPL3D, the authors aim at animation generation from natural language input.

XSAMPL3D was also influenced by the field of animation scripting languages. For instance, the Alice3D scripting language [PAB+97] aims at enabling 3D animation authoring

for novices. As in XSAMPL3D, 3D object manipulations (or actions) can be described in an intuitive manner. For instance, objects can be moved to an absolute position or displaced relative to their current position. As other animation scripting languages, Alice3D contains elements for the definition of synchronization aspects, such as parallel and sequential movements. However, Alice3D does not explicitly focus on hand-object interaction and virtual character animation.

The syntax and semantics of XSAMPL3D synchronization elements, such as Parallel and Sequential, were primarily inspired by SMIL, the Synchronized Multimedia Integration Language [Wor98], while the reuse of time containers is geared to the DEF/USE-mechansim in X3D or VRML.

6 Conclusion

In this paper, we proposed a new approach for the representation of actions involving object manipulations and their reproduction by virtual humans, which is part of the action capture overall process described in [JAHW06]. Actions are encoded in a domain-specific XML-dialect called XSAMPL3D. XSAMPL3D offers a compact and human-readable notation at a high-level of abstraction. XSAMPL3D descriptions can be either derived from recorded interaction data or created manually for rapid prototyping of animations. XSAMPL3D documents are automatically translatable into XML behavior plans, which provide a solid foundation for the animation of virtual characters.

As a first proof-of-concept, the approach was successfully applied by students in a practical course. In this course several scenarios were realized, such as different kitchen scenarios (e.g. the example described in this paper) and an office desk scenario.

We plan to evaluate the approach in further scenarios, especially scenarios including the operation of control actuators. Since the existing physics support for control actuators has been implemented and tested using a separate framework, this functionality has to be integrated as a component into our current virtual human animation system. In addition, the flexible hierachical structure of XSAMPL3D facilitates the extension with elements that allow the description of more than two parallel (i.e. left and right hand) activities. Extension possibilities comprise the integration of head and foot movements or multiple (virtual) humans.

References

[Arb02] M. A. Arbib. The mirror system, imitation, and the evolution of language. In *Imitation in Animals and Artefacts*. MIT Press, 2002.

[BADV+08] H. Ben Amor, M. Deininger, A. Vitzthum, B. Jung, and G. Heumer. Example-based synthesis of goal-directed motion trajectories for virtual humans. In *5. Workshop "Virtuelle und Erweiterte Realität*. GI-Fachgruppe VR/AR, 2008.

[BAHJV08] H. Ben Amor, G. Heumer, B. Jung, and A. Vitzthum. Grasp synthesis from low-dimensional probabilistic grasp models. *Computer Animation and Virtual Worlds*, 19, 2008.

[Gle98] M. Gleicher. Retargetting motion to new characters. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 33–42, New York, NY, USA, 1998. ACM.

[Gom08] Frank Gommlich. Simulation bewegbarer Norm-Stellteile in dynamischen virtuellen Umgebungen. Master's thesis, TU Bergakademie Freiberg, 2008.

[JAHW06] B. Jung, H. Ben Amor, G. Heumer, and M. Weber. From Motion Capture to Action Capture: A Review of Imitation Learning Techniques and their Application to VR-based Character Animation. In *Proceedings VRST 2006 - 13th ACM Symp. on Virtual Reality Software and Technology*, pages 145–154, 2006.

[Ken04] A. Kendon. *Gesture:Visible Action as Utterance*. Cambridge Univ. Press, 2004.

[KKW02] A. Kranstedt, S. Kopp, and I. Wachsmuth. Murml: A multimodal utterance representation markup language for conversational agents. In *Proc. of the AAMAS Workshop on Embodied conversational agents*, 2002.

[MMK06] M. Ma and P. Mc Kevitt. Virtual human animation in natural language visualisation. *Artif. Intell. Rev.*, 25(1-2):37–53, 2006.

[PAB+97] J. S. Pierce, S. Audia, T. Burnette, K. Christiansen, D. Cosgrove, M. Conway, and K. Monkaitis et al. Alice: Easy to use interactive 3d graphics. In *ACM Symp. on User Interface Software and Technology*, pages 77–78, 1997.

[Sch19] G. Schlesinger. Der Mechanische Aufbau der Künstlichen Glieder. In M. Borchardt et al., editors, *Ersatzglieder und Arbeitshilfen für Kriegsbeschädigte und Unfallverletzte*, pages 321–661. Springer-Verlag, Berlin, Germany, 1919.

[VCC+07] H. H. Vilhjalmsson, N. Cantelmo, J. Cassell, N. Ech Chafai, M. Kipp, S. Kopp, and M. Mancini et al. The behavior markup language: Recent developments and challenges. In *Proc. 7th International Conference on Intelligent Virtual Agents, IVA-2007*, pages 99–111. Springer, 2007.

[WHAJ06] M. Weber, G. Heumer, H. Ben Amor, and B. Jung. An animation system for imitation of object grasping in virtual reality. In *Proceedings of the 16th International Conference on Artificial Reality and Telexistence*. Springer, 2006.

[Wor98] World Wide Web Consortium. *Synchronized Multimedia Integration Language (SMIL) 1.0 Specification*, 1998.