XSAMPL3D: An Action Description Language for the Animation of Virtual Characters

Arnd Vitzthum[*], Heni Ben Amor[*], Guido Heumer[*], Bernhard Jung[*]

[*]Virtual Reality and Multimedia Group
Institute for Informatics
Technical University Bergakademie Freiberg
{vitzthum, amor, heumer, jung}@informatik.tu-freiberg.de
vr.tu-freiberg.de

Abstract

In this paper we present XSAMPL3D, a novel language for the high-level representation of actions performed on objects by (virtual) humans. XSAMPL3D was designed to serve as action representation language in an imitation-based approach to character animation: First, a human demonstrates a sequence of object manipulations in an immersive Virtual Reality (VR) environment. From this demonstration, an XSAMPL3D description is automatically derived that represents the actions in terms of high-level action types and involved objects. The XSAMPL3D action description can then be used for the synthesis of animations where virtual humans of different body sizes and proportions reproduce the demonstrated action. Actions are encoded in a compact and human-readable XML-format. Thus, XSAMPL3D describtions are also amenable to manual authoring, e.g. for rapid prototyping of animations when no immersive VR environment is at the animator's disposal. However, when XSAMPL3D descriptions are derived from VR interactions, they can accomodate many details of the demonstrated action, such as motion trajectiories, hand shapes and other hand-object relations during grasping. Such detail would be hard to specify with manual motion authoring techniques only. Through the inclusion of language features that allow the representation of all relevant aspects of demonstrated object manipulations, XSAMPL3D is a suitable action representation language for the imitation-based approach to character animation.

First presented at 6th Workshop 'Virtuelle und Erweiterte Realität', GI-Fachgruppe VR/AR 2009, extended and revised for JVRB

Keywords: Virtual Humans, Action Representation, Imitation-Based Animation

1 Introduction

Creating animations of virtual humans that realistically grasp and manipulate objects is a highly complex task. Subproblems to be solved include the generation of suitable hand shapes that result in stable grasps of the manipulated object. Also, natural appearing motions of arms and hands must be synthesized for the different phases (i.e. hand-object approach, grasping, manipulation, and retraction) of each action. For the synthesis of such motions a number of parameters such as the grasp type or the grasp location need to be determined. This often depends on the intended kind of manipulation (e.g. hand shapes when grasping a knife will be very different when cutting something as opposed to putting it into the drawer). Further, for bimanual manipulations, movements of the left and right hand must be synchronized.

Due to the high degree of articulation of the arm-hand-finger system purely manual animation editing or adaption of motion capture data quickly becomes

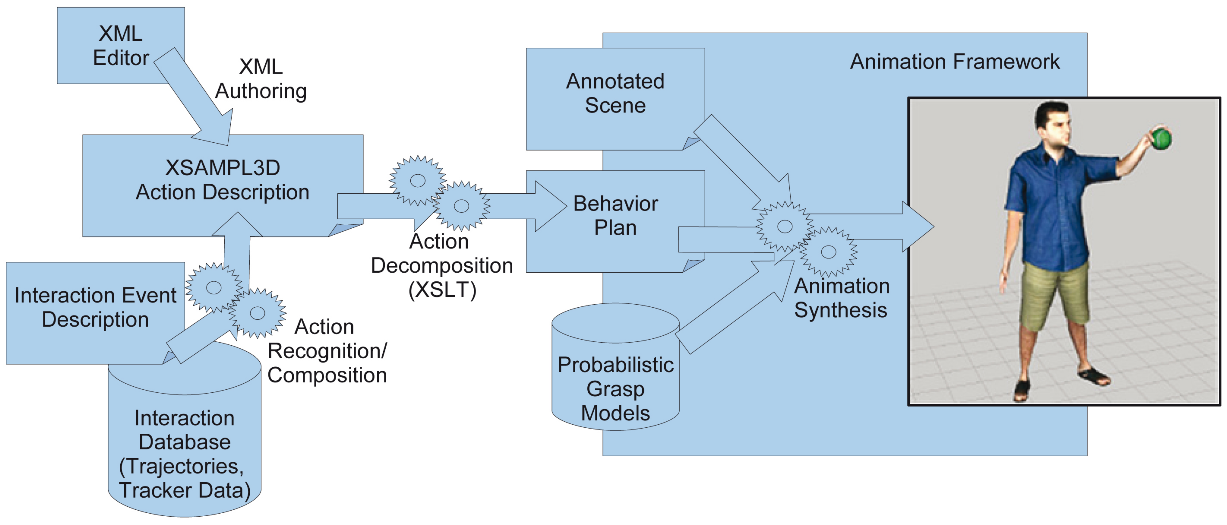
XML Editor
XML Authoring
XSAMPL3D Action Description
Interaction Event Description
Interaction Database (Trajectories, Tracker Data)
Action Recognition/ Composition
Action Decomposition (XSLT)
Annotated Scene
Behavior Plan
Probabilistic Grasp Models
Animation Framework
Animation Synthesis

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

a very labor-intensive task. The problem is especially severe when several variations of essentially the same manipulation sequence are to be created. An example are virtual prototyping settings where animations of many different virtual humans need to be generated in order to test the ergonomics of machine operating procedures. For each virtual human or other small difference in the target situation, the generated motion needs to be adapted anew, i.e. the well known retargetting problem [Gle98]. Better suited for such purposes are model-driven approaches that synthesize animations from high-level specifications of the action (e.g. [RG91, DLB96, KL00, YKH04, ACT05]). The automatic adaption to the target scenario is achieved via different "intelligent" functionalities e.g. for determining object-specific grasp positions and types (e.g. [DLB96, KT99]), collision free reaching [KAAT03] and object displacement motions [KKN$^+$02]. Besides the complexities of implementing such intelligent functionalities, a problem is that generated motions often appear "robotic" as compared to more natural looking motion capture-based methods. Further, the designer has relatively little control over the details of the automatically generated animations.

Our work generally addresses the problem of how animations of complex manipulation tasks can be specified such that easy adaptation to new situations is possible but, at the same time, the animator is given given fine control over movement details. For this, we have developed an imitation-based approach that makes use of VR techniques [JBAHV11]: 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 is abstracted to a high-level *action representation*. Finally, animations of virtual characters are synthesized 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 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*, *displace*, and *release* which are then combined to complete actions on the involved object. XSAMPL3D representations are also compact enough to be directly authored by an animation designer (upper left corner in Figure 1), e.g. for rapid prototyping of animations showing virtual humans performing object manipulations.

2 Related Work

The most important elements of XSAMPL3D are actions, i.e. goal-directed movements. A complete XSAMPL3D description is composed of action sequences. An action in XSAMPL3D describes what is done with an object at an abstract level (e.g., turn cup), rather than exactly defining how this is achieved in a VR environment by moving the limbs of a virtual character. For this reason, XSAMPL3D should not be considered as an animation language, but as a domain specific language for the description of actions on objects. From this point of view, there is only few research work closely related to XSAMPL3D. However, although XSAMPL3D is not an animation language per se, it is possible to map XSAMPL3D actions to animations of virtual humans, as described later in detail. Therefore we also investigated animation languages for virtual humans during the design of XSAMPL3D.

One approach which aims at similar scenarios as XSAMPL3D is the Parameterized Action Representation (PAR) [BEL02]. As in XSAMPL3D, PAR can be used to animate virtual characters which perform actions on objects. Instead of XML, in this approach natural language processing is applied in order to combine virtual humans, actions and objects. Actions are organized in a tree structure and are executed in depth-first order. Non-leafs represent more complex composed actions while leaves of the tree represent low-level actions such as reaching a certain object. XSAMPL3D action descriptions have a similar structure, but don't contain low-level actions as PAR. For example, there is no need to specify a reach action explicitly, since a reach must always be performed in order to manipulate an object. This high level of abstraction allows a very compact and readable action description in XSAMPL3D. In PAR, actions can be stored in an action dictionary (called Actionary) for reuse. Reuse of (composed) actions is also possible in XSAMPL3D, although another mechanism is used, inspired by the DEF/USE-mechanism in X3D or VRML.

The CONFUCIUS system introduced in Ma and McKevitt [MM06] 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 action components. In contrast to XSAMPL3D and similar to PAR, CONFUCIUS aims at animation generation from verbal instructions instead of from physical demonstration.

Many research efforts in virtual human animation went into the field of Embodied Conversational Agents (ECAs). Languages and systems in the field of ECAs are e.g. the Multimodal Utterance Representation Markup Language (MURML) [KKW02], the Avatar Markup Language (AML) [KMTA$^+$02], the Rich Representation Language [PKS$^+$02], the Behavior Markup Language (BML) [VCC$^+$07] and the Player Markup Language [Jun08].

Although in general these languages focus more on communicative behavior such as gestures, facial expression, gaze and speech than on the interaction with virtual objects, they share some common aspects with XSAMPL3D, such as action compositing, the synchronization and timing of activities (the concrete syntax and semantics of XSAMPL3D synchronization elements, such as Parallel and Sequential, were primarily inspired by SMIL, the Synchronized Multimedia Integration Language [Wor98], a language which was developed for the synchronization of different digital media, such as audio, images, video and text). In contrast to these systems, the virtual humans in our animation framework are no "intelligent" agents, since they are not able to react to events such as user interactions; they simply execute a predefined sequence of actions. However, for our target application scenarios (ergonomic evaluations) this deterministic behavior is completely sufficient and significantly reduces the complexity of the overall system. Virtual humans of different proportions and sizes can be used to perform the same tasks in VR in order to identify ergonomic problems (e.g., if objects are out of reach or if the Virtual Human's arm collides with an object while trying to grasp another object).

An 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.

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.

3 VR Infrastructure for Imitation-Based Character Animation

Even though the action language presented here can also be used to rapid-prototype animations by manually creating action 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. These demands are briefly described in the following section; for a more detailed account see [JBAHV11].

3.1 Scene Representation

An important component of an action description and animation system is formed by the objects on which actions are performed. 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 [WHBAJ06]; for an extension to articulated objects such as control actuators like knobs, switches etc. see [GHJ+12]. Object type descriptions are written in an XML-based representation structure and stored in a common database for reference.

3.2 Interaction Recording

Since our approach for animation synthesis is primarily based on imitation learning there needs to be some way to record VR interaction data. Most importantly, *arm trajectory* and *hand posture* data during grasping is 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 constituent 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 during hand-object contact is automatically classified w.r.t. its grasp type [HBAJ08]. During the course of the virtual workers project several grasp taxonomies from the medical, e.g. [Sch19], and robotics literature, e.g. [Cut89], have been explored. Since some significant shortcomings in relation to applicability in virtual environments have been identified, additionally a new taxonomy has been proposed in [Heu10] which as a unique feature provides support for different types of non-prehensile grasps. The latter play an important role in the operation of control actuators in application domains like virtual prototyping, ergonomics evaluations and simulation of machine operation procedures.

The result of the intermediate interaction analysis process is the detection of basic interactions (like reach-motions, grasping, releasing and manipulations of objects) which are propagated to other application parts as *interaction events*. 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. An in-depth discussion of this process can be found in [Heu10].

4 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). Besides being processable by machines this representation of-

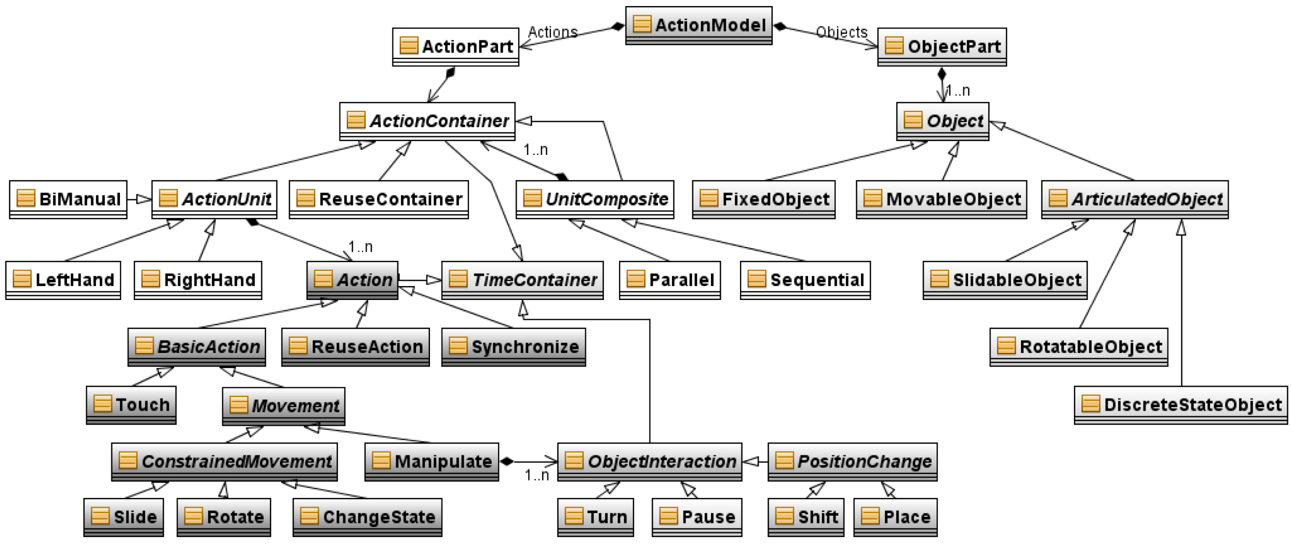

Figure 2: Simplified overview of the XSAMPL3D schema elements.

fers a *human-readable syntax* at a *high level of abstraction*.

As mentioned before, XSAMPL3D descriptions can either be manually created or automatically generated from interaction events (see Section 3.2). Both creation methods provide advantages in special use cases.

Manual creation of XSAMPL3D action descriptions enables *rapid scripting of animations* for testing purposes, since descriptions can be mapped to animation specifications later (see Section 4.4).

In the second creation scenario (*automatic creation*), previously captured action sequences that were encoded into a lower level data representation (such as interaction events, trajectories and motion records), can be presented in a human readable form after transforming them into XSAMPL3D action descriptions. This facilitates *post-editing* as well as the *identification and removal of erroneous data* (e.g., caused by recording errors). Because of its high level of abstraction, an XSAMPL3D description does not contain all of the details contained in the original data. 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 fidelity*: From a "pure" XSAMPL3D description (i.e, the animation player ignores links to lower-level data), an animation can be generated which reflects the main features of a previously recorded human motion (low level of fidelity). For instance, captured motion trajectories will be replaced by simple calculated trajectories while the overall timing will be preserved. If, however, links to interaction event data are taken into account, it is possible to reproduce the original motion very precisely (high level of fidelity).

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 is 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 (slightly 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

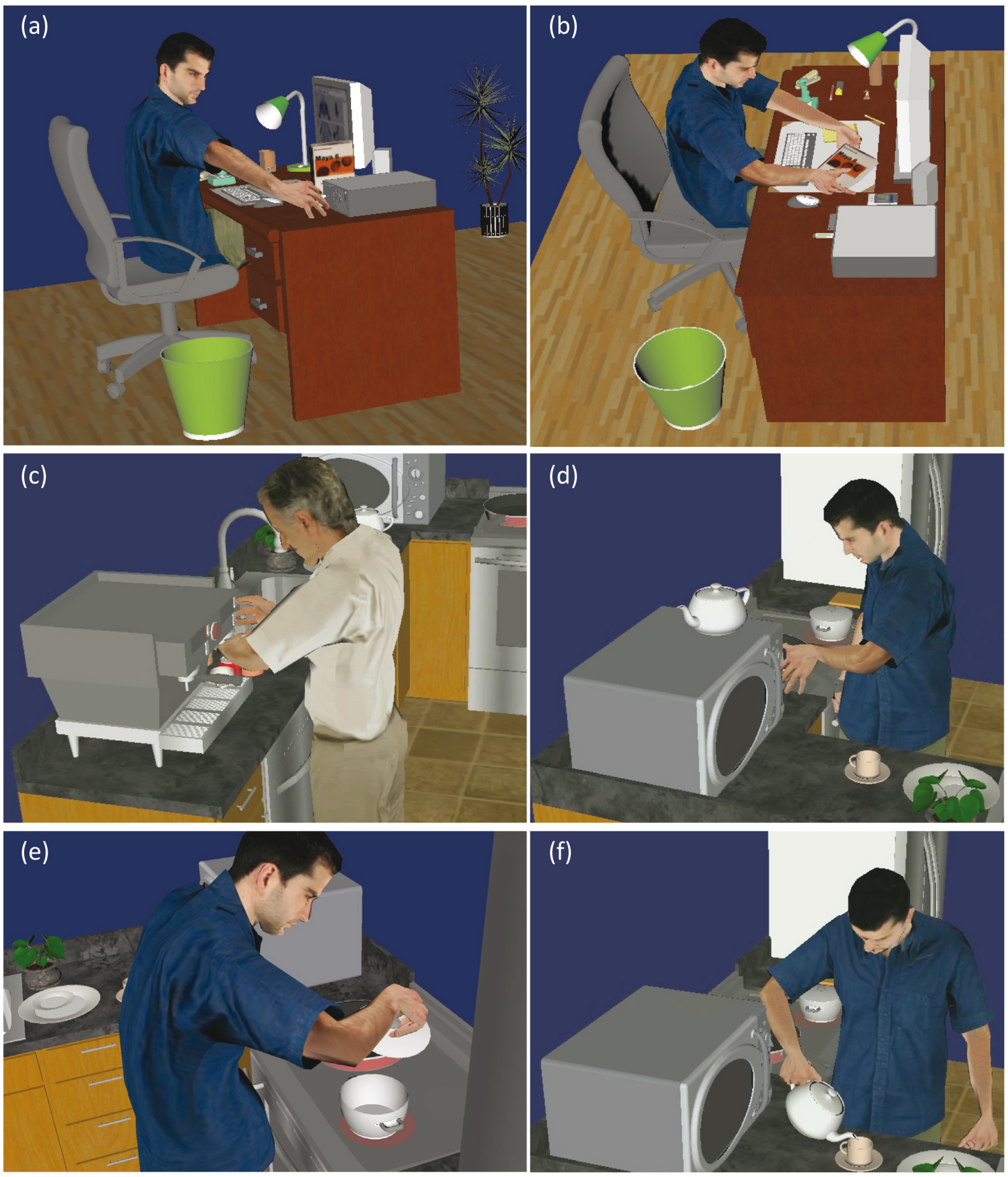

Figure 3: Different scenarios developed with XSAMPL3D. (a) Plugging in a USB memory stick. (b) Moving a book bimanually. (c) Preparing a cup of espresso. (d) Turning on a microwave. (e) Lifting a pot's lid. f) Pouring a cup of tea.

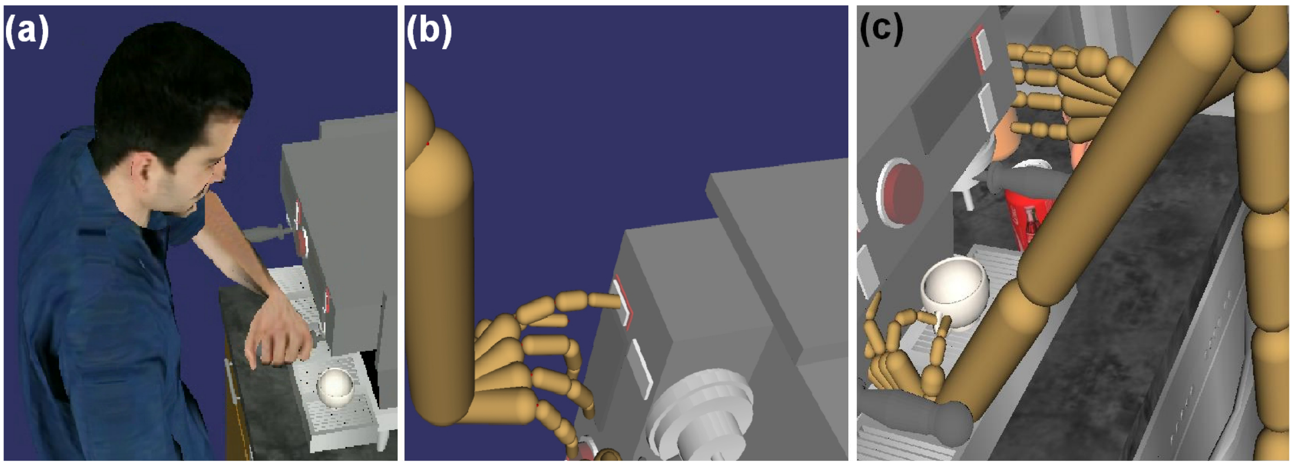

Figure 4: Espresso machine example. (a) The virtual human attaches a portafilter to an espresso machine. (b) The virtual human (skeletal representation) presses a button to fill a cup. (c) The cup is taken out of the machine.

primarily address in our current applications, such as ergonomic studies on virtual 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.

4.1 Example

In order to show the feasibility of our approach, XSAMPL3D was applied to the development of different example scenarios in a practical course together with students at the Technical University Freiberg. These scenarios take place in a office environment (figure 3(a) and (b)), a kitchen environment (figure 3(c)–(f)) and a car environment (figure 16). In the following, we explain the capabilities of XSAMPL3D in more detail by one selected brief scenario (Figure 4).

4.2 Action Specification

The XSAMPL3D action specification includes different aspects, such as objects, action types, action composition, synchronization and timing.

Objects

Objects can be divided into several classes: *fixed objects*, *movable objects* and *articulated objects* (e.g. control actuators). Articulated objects have specific movement constraints which are defined in a separate *annotated object document* (see Section 3). 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).

Actions

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

```
<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>
```

Figure 5: Declaration of the action `AttachFilter` in XSAMPL3D.

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

Figure 6: Example action unit containing a sequence of three actions.

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 similar 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 an angle and an optional axis (the "Up"-axis of the object by default). For example, in the kitchen scenario, the portafilter is first moved to the espresso machine and attached to it afterwards by a turn. The corresponding action description is shown in Figure 5.

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 when 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 shown in Figure 5 it will take 0.5 seconds to turn the portafilter. The sum of all interaction durations and the reach duration is equal to the overall action duration.

Action Composition

Actions can be grouped together using an *action unit*. An action unit contains a sequence of actions which are performed with the same hand or with two hands cooperatively (i.e. *bimanually*). W.r.t. the classification of bimanual activities proposed by Guiard [Gui87], a bimanual action in XSAMPL3D models a *symmetric* activity performed with both hands *in phase*. 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 in Figure 6 specifies that all actions of the kitchen scenario are to be performed consecutively with the left hand (simplified).

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 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.

Synchronization and Timing

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 sequential composite ends if the last contained element (in terms of order)

```
<Sequential>
  <Parallel>
    <LeftHand startTime='0' relaxDuration='1' ID='U1'>
      <!--Attach filter, duration 2 seconds-->
      <Manipulate ... object='PortaFilter'>...</Manipulate>
    </LeftHand>
    <RightHand startTime='2' relaxDuration='1' ID='U2'>
      <!--Press button, duration 2 seconds-->
      <Touch ... object='Button'/>
    </RightHand>
  <Parallel>
  <LeftHand startTime='2' relaxDuration='1' ID='U3'>
    <!--Take cup, duration 2 seconds-->
    <Manipulate ... object='Cup'>...</Manipulate>
  </LeftHand>
</Sequential>
```

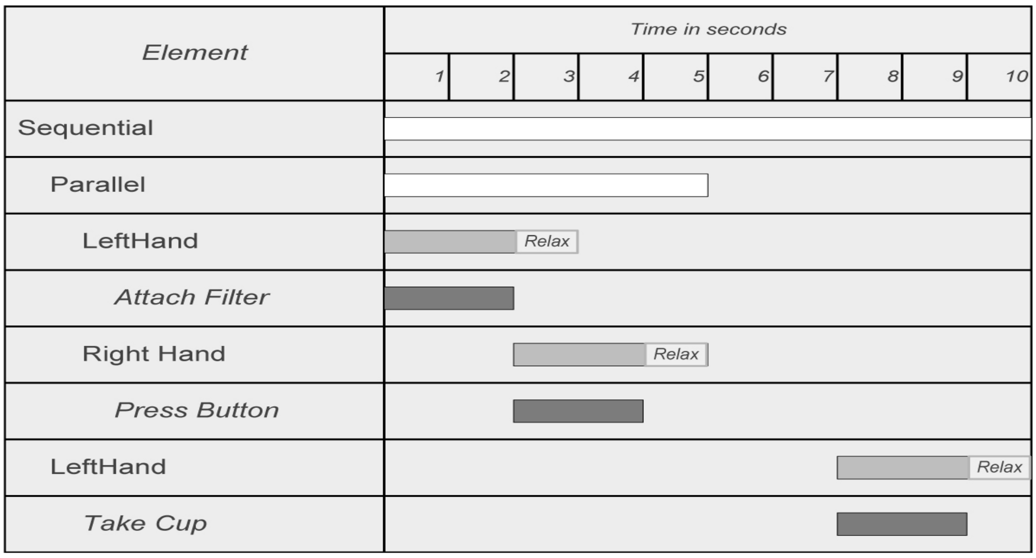

Figure 7: An example of nested unit composites (upper part: code view, lower part: timeline view). First the `AttachFilter` action is performed using the left hand (unit `U1`). While the left arm is still in its relaxation phase, the right arm already starts to press the espresso machine button (unit `U2`). Since `U2` ends last, the `Parallel` composite ends with this unit. After waiting two seconds (start time of `U3`), the cup is taken out of the machine.

ends. A parallel composite ends when the element that ends last (in terms of time) 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 attaching the portafilter) 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, i.e. one can define parallel unit composites which are contained in a sequential composite and vice versa (see Figure 7). To realize such a containment hierarchy, the composite design pattern was applied.

Object interactions, 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 *Attach-Filter*-action defined in Figure 5.

A 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),

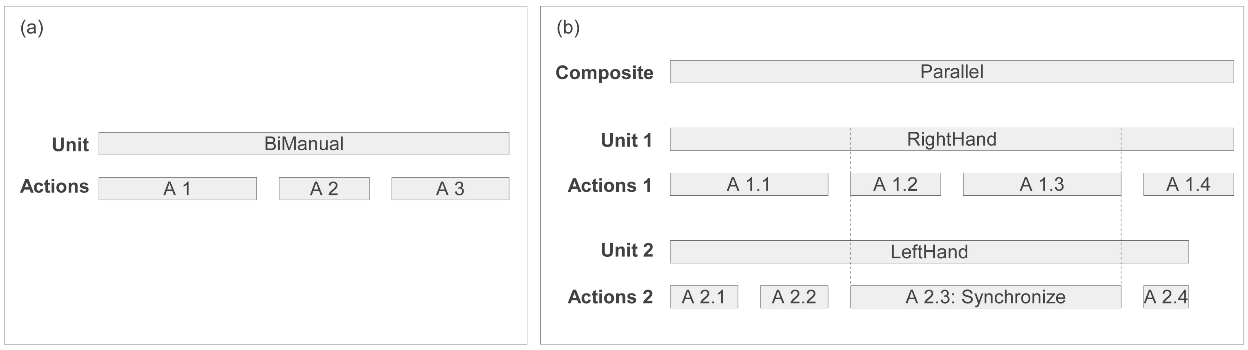

Figure 8: (a) A `BiManual` unit describes a sequence of actions which are performed bimanually. (b) A `Synchronize` action can be used to ”synchronize” the movements of one hand with the movements of the other hand in order to perform selected actions of parallel sequences with both hands. In the example, actions 2 and 3 of unit 1 are performed bimanually.

`Synchronize` also allows for partially synchronizing parallel units (Figure 8). 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. In this way, a synchronize action enables a finer control of symmetric bimanual activities.

4.3 Action Recognition and Composition

As noted in the introductory sections, XSAMPL3D descriptions can be automatically derived from user interactions in VR. In this section, we describe the process of transforming sequences of intermediate-level interaction events (see Section 3) into high-level XSAMPL3D action descriptions. Since recorded interaction event sequences can be stored as XML documents, we have chosen the powerful transformation standard XSLT (version 2.0) to implement transformation rules. An advantage of this approach is the opportunity to apply existing XSLT-engines. In this case, we used the schema-aware Saxon engine (www.saxonica.com) with some custom Java-extensions. For the sake of completeness it should be mentioned here that our interaction recording framework also provides mechanisms for the integration of *live* action recognition, i.e. the recognition of actions during a recording session. However, live action recognition has not been implemented yet.

Interaction events in an XML input document are organized in one or more subsequences. Each subsequence normally contains a list of events which occurred close together in time and thus usually represent a series of related actions. Beside a timestamp (`start-time`), each event contains additional information, such as ID, event type and applied grasp type, object name, duration, but also lower level information, such as finger joint angles, transformation of the hand in global (world) coordinates, transformation of the hand in local (object) coordinates, etc. The concrete data contained in an event depends on the type of the event. For instance, a `grasp` event contains a grasp type, while a `reach` event does not provide such information. Hand coordinates and other lower-level data are included for both the start and the end time (sum of `start-time` and `duration`) of an event (`start-data` and `goal-data`). Trajectories are not directly included in an event specification that involves hand movement, but are stored in the interaction database and referenced via IDs. As an example, the event code in Figure 9 describes the displacement of a bottle held in a cylindrical grasp.

Figure 10 illustrates single steps of the transformation process. This process only applies for free object manipulations, since constrained object movement recognition requires consideration of additional events (such as state change events of control actuators, e.g., `ButtonPressed` events) and is – as already mentioned – out of the scope of this paper.

In the first transformation step (*Reduce Data*), data not required for action recognition is removed from the input document. For instance, raw sensor data of finger joint angles can be disregarded (finger joint angles are particularly important for grasp recognition, which already takes place at the interaction recognition level).

In the second step, the *Basic Structure* of the target document is created. Information about all manip-

```
    <event type="displace" start-time="2.29079" duration="0.804834"
        id="IE-184ff9a4-0f55-11df-bef6-00142225ef0d">
      <references>
        <ref type="hand-trajectory" id="MET-184eec58-0f55-11df-8a15-00142225ef0d"/>
      </references>
      <low-level-features>
        <hand-side>right</hand-side>
        <start-data>
          <hand-posture>
            <joint-angle joint-id="r_index1">...</joint-angle>
            ...
          </hand-posture>
          <hand-transform-wcs>...</hand-transform-wcs>
          <hand-transform-ocs>...</hand-transform-ocs>
          ...
        </start-data>
        <goal-data>...</goal-data>
      </low-level-features>
      <high-level-features>
        <target-object>Bottle-1</target-object>
        <grasp-type taxonomy="Schlesinger">cylindrical</grasp-type>
      </high-level-features>
    </event>
```

Figure 9: Example code of a `displace` event (simplified).

ulated objects is extracted from the interaction event sequence and stored in the `Objects` description part of the target XSAMPL3D document (see Figure 2). Beside the `Objects` part, the `Actions` part is generated in this step, which will later contain the actual action specifications. Each subsequence of events is translated into a `Parallel` container, as it can contain events representing simultaneous left- and right-handed movements.

In the third step (*Separate Events*), for each parallel container left- and right-hand events are separated from each other by forming a left hand unit and a right hand unit. In the next step (*Reduce Events*), events which obviously occurred due to recording inaccuracies are removed. For example, tracker jitter can cause a sequence of fast alternating `release` and `grasp` events, which can be deleted.

In step 5 (*Find Actions*), patterns of events are identified that represent actions on objects. As already described in Section 4.2, an action is typically represented by the following pattern: one or more `reach` events, a `grasp` event, zero or more `displace` events and a `release` event (in this order). E.g. for freely movable objects, an action can either be a touch (if the object is not displaced) or manipulate action. For each action, the IDs of the events from which it is composed are stored in an (optional) attribute. Thus, it is possible to retrace the mappings between events and actions.

In the next step (*Classify Interactions*), the object interactions of each manipulate action are classified using the Java-based machine-learning toolkit WEKA [WF05]. As a part of an action, an interaction comprises a sequence of displace events. All trajectories referenced by these events are connected with each other in order to form the "overall trajectory" of the interaction. The classification is based on features derived from this overall trajectory, such as the *trajectory length* after projection to the horizontal plane and the *trajectory height*. The classification also depends on the applied grasp type and on the amount an object was rotated during manipulation. For instance, a `Turn` interaction is normally characterized by a rather short and "flat" trajectory (small values of trajectory length and height) and a rather large rotation amount. In addition, it is performed using a prehensile grasp.

Afterwards, attribute values for actions and interactions are derived (*Calculate Attribute Values*). As an example, the duration of an interaction can be derived by subtracting its start time (`start-time` of the first event that belongs to the interaction) from its end time (`start-time` plus `duration` of the last event that belongs to the interaction). Another example is the calculation of the amount an object was rotated during

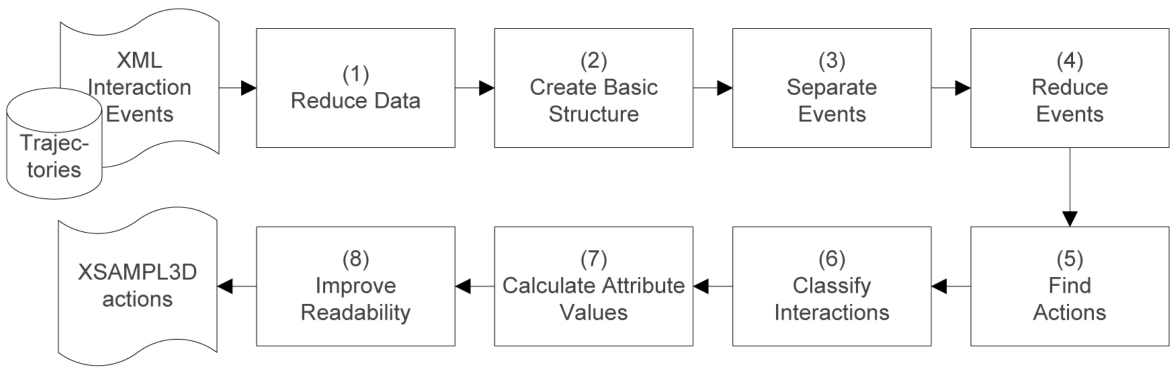

Figure 10: The transformation of interaction events into actions.

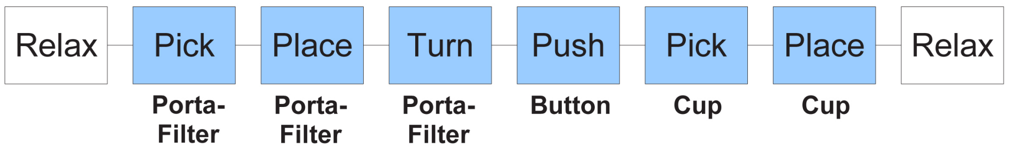

Figure 11: A sequence of behaviors used to create an animation in which a virtual human brews a cup of coffee.

a turn interaction (in axis/angle representation) from recorded matrices. Finally, the readability of the target document is improved, e.g., by removing unnecessary white spaces (*Improve Readability*).

4.4 Action Decomposition

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). As for the transformation of interaction events into actions, we used the schema-aware Saxon-XSLT-engine, which also allows us to access type information from the XSAMPL3D schema in XSLT. If necessary, the generated XML-plans can be manually refined.

5 Behavior Language and Execution

In our architecture, actions describe the animation at a high-level of abstraction. A lot of information about *how* the action is executed in detail is 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 rotations 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 (see Section 5.1 for more detail).

The behaviors contained in the framework include among others *relax*, *push*, *pick*, *place* and *turn* behaviors. The *relax* behavior makes the virtual human take on a relaxed body posture. This is useful for chaining several action 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

```
<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>
        <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>
```

Figure 12: Instatiation of a Pick behavior.

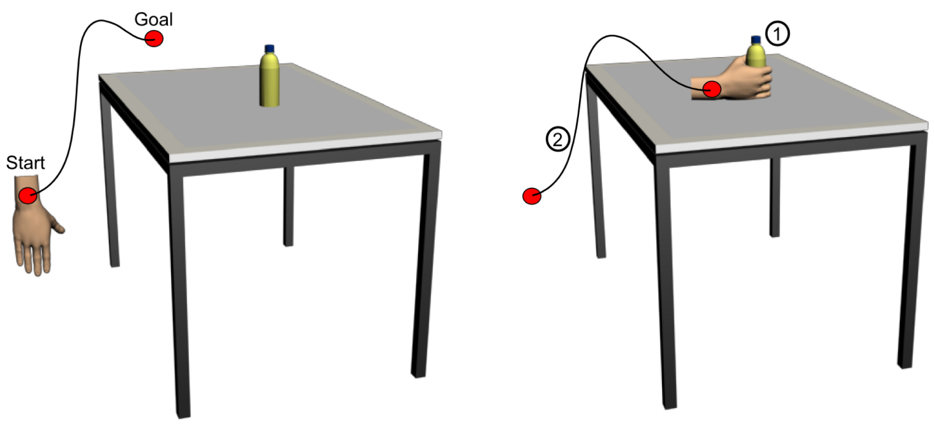

Figure 13: Left: The starting point of the grasping algorithm. The goal is to grasp the object by moving along a recorded trajectory. Right: First a suitable grasp is found using optimization (1). Then, the new position of the wrist is used to retarget the trajectory (2).

of behaviors provided in an XML representation. Figure 11 shows the behavior sequence for the kitchen scenario. Each behavior supported by the framework can be instantiated and parameterized in XML. The XML code in Figure 12 shows the instantiation of a `Pick` behavior.

Apart from the start and end time of the behavior, the user can also supply the name of the object 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.

5.1 Grasping Strategy

Our motion synthesis algorithm for grasping behaviors has been described in more detail in [JBAHV11]. In the following we will therefore only roughly outline the main steps of the grasping algorithm. In particular we will show how the algorithm ensures that even new objects or displaced objects can be correctly grasped.

The synthesis of a new grasping animation consists of two steps, as can be seen in Figure 13. 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. Probabilistic hand shape models learnt from 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 [BAHJV08]. While

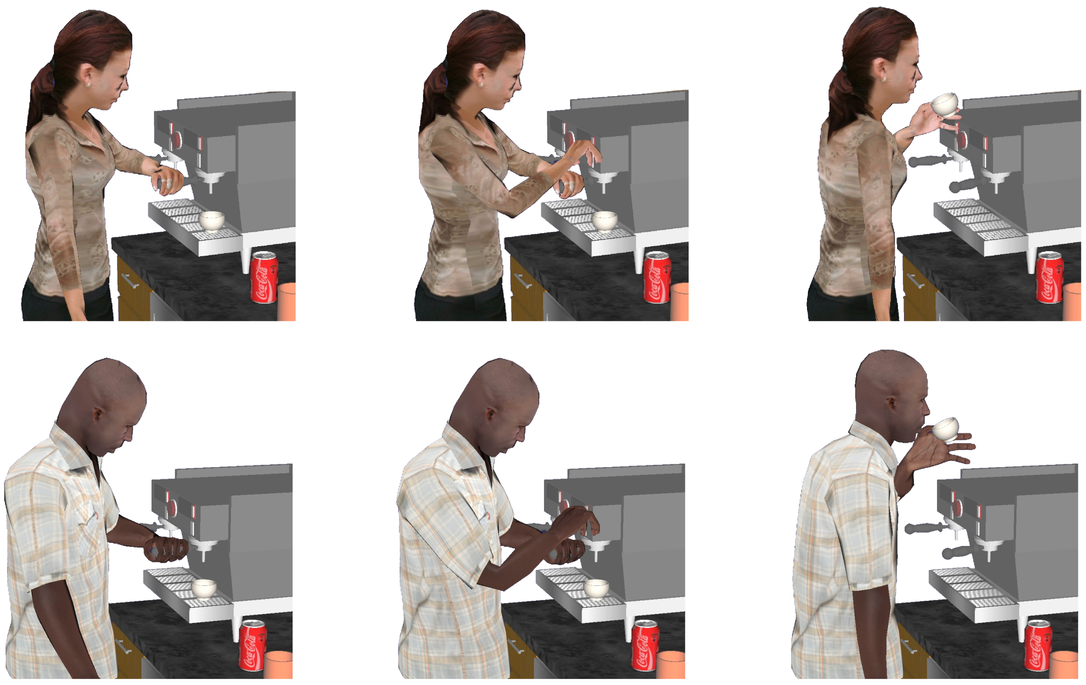

Figure 15: Sequence of frames from animations synthesized for two different virtual humans. The animations are synthesized from the same action description. The behavior-based animation framework ensures that the animations are retargeted to the current virtual human.

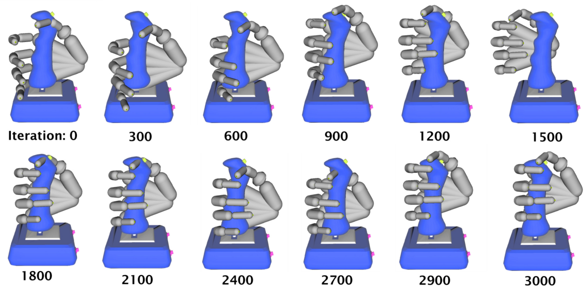

Figure 14: During grasp optimization, candidate grasps are generated from probabilistic hand shape models [BAHJV08]. If present in the XSAMPL3D action description, hand shape, wrist position and wrist orientation from a demonstrated VR interaction can be used to seed the optimization process.

the hand shape, wrist position, and wrist orientation of a grasp demonstrated during a VR interaction cannot be directly applied to the animated character (amongst other reasons, due to difference in body and hand size), such information can still be used as initial seeds in order to speed up the optimization process (see Figure 14 for a visualization of the optimization process starting from an initial seed).

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 in a hand-centered coordinate system. 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 architectures, such as the H-Anim format, allow us to replay the animation with a number of available virtual humans.

6 Discussion

XSAMPL3D action descriptions allow the specification of goal-oriented behavior independently of the body sizes and proportions of the animated virtual humans. E.g., Figure 15 shows two different virtual humans that are animated from the same XSAMPL3D document. As can be seen from this figure, the animations are synthesized based on the anatomical proper-

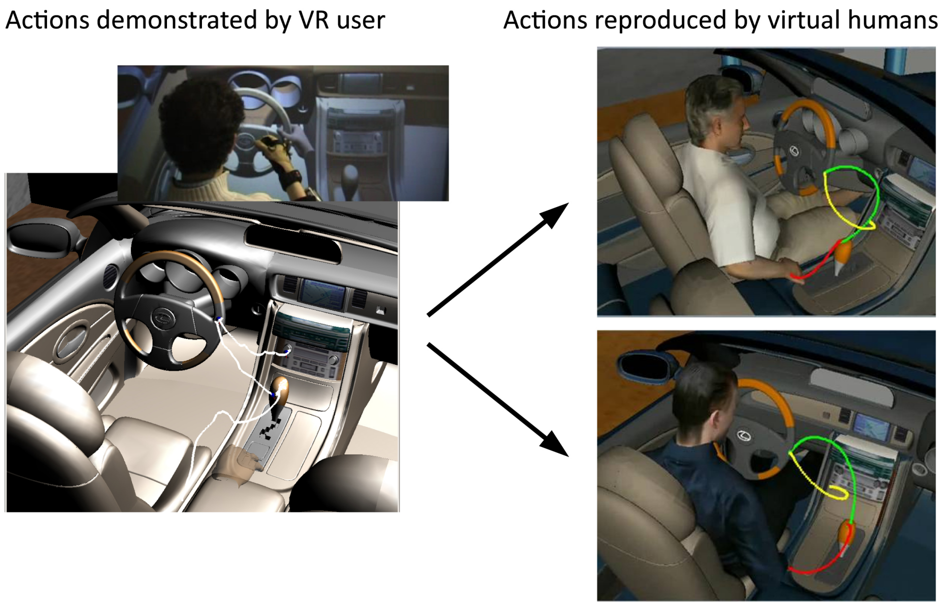
Actions demonstrated by VR user
Actions reproduced by virtual humans

Figure 16: Left: A user performs several actions in a cockpit scenario. The motion is shown as a white trajectory. Right: Recorded action files are used to animate virtual humans of different size and body proportions. Animations are robust against changes in the virtual prototype; e.g. in the lower image, the gear-shift has been repositioned. Colored lines indicate the trajectories of the right hand's wrist while interacting with several control elements in a car.

ties of the virtual humans, i.e. the respective target objects are reached despite different sizes and body proportions of the characters. As another example, Figure 16 shows two virtual humans performing the same sequence of actions in a car interior. Again, both animations were automatically generated from the same behavior specification. In the second animation, however, the stick shift's position was slightly modified as compared to the first animation. This demonstrates that the XSAMPL3D high-level animation specifications are also robust against certain modifications of the virtual environment. Of course, such modifications to the virtual enviroment should remain within reasonable limits. E.g., if the stick shift was displaced by a too large margin or if a small child was to perfom the specified action sequence, only very unnatural looking or even no valid animations could be generated with the presented approach.

XSAMPL3D action descriptions can either be manually authored or automatically derived from action sequences demonstrated by a user of an immersive VR environment. In the latter case, action descriptions can accomodate fine movement details from the recorded VR interaction. However, care must be taken in order not to repeat overly cautious movements by the VR user in the resulting animations of the virtual humans. E.g., the left of Figure 16 shows the trajectory of a VR user's wrist position while interacting with several control elements of a car. The jitter in the recorded trajectory can be explained by possible inaccuracies of the tracking devices and, more importantly, the missing haptic feedback during the interaction which causes rather cautious, somewhat unnatural movements of the VR user when operating the virtual prototype. By abstracting to high-level action descriptions, such imperfections of the demonstrated actions can be compensated for. In the shown example, smooth animation trajectories are generated by adapting suitable prerecorded movements from a trajectoty database. Alternatively, the jittery trajectory of the VR object manipulation could be smoothed and adapted to the target enviroment. Similarly, the timing of the reproduced animation may differ from the original action demonstration and can be varied to make the generated animations appear more natural.

Our animation synthesis implementation currently does not include sophisticated path planning techniques for the generation of end-effector trajectories. The underlying assumption is that the original action is typically performed such that no unintended collisions/penetrations of scene objects occur. Therefore imitation of the original action is also likely to be collision-free. To account for situations where this assumption does not hold, animation synthesis could be combined with a dedicated path / motion planner such as [KKN+02, KAAT03].

Further, our inverse kinematics algorithm is currently purlely model-driven. Our approach could alternatively make use of example-based inverse kinematics techniques such as [GMHP04]. Similar to the hand shape optimization process described in Section 5, arm joint configurations recorded during a VR interaction could possibly provide useful information to speed up optimization calculations for such example-based inverse kinematics algorithms.

7 Conclusion

We developed a new action description language suitable for the synthesis of virtual character animations involving object manipulations. Actions are encoded in a domain-specific XML-dialect called XSAMPL3D. In contrast to motion capture data that merely describes posture changes over time, the key idea is to represent actions in terms of high-level features such as the object being manipulated, grasp type etc. Thus, action representations are valid for differently sized virtual humans which avoids the need for manual (and labor-intensive) retargetting of the conventional motion capture approach. XSAMPL3D offers a compact and human-readable notation at a high level of abstraction. Within our action capture framework, XSAMPL3D descriptions can be either created manually for rapid prototyping of animations or automatically derived from recorded VR interaction data. The latter case provides a fast means for the generation of XSAMPL3D descriptions (if an immersive VR installation is available). XSAMPL3D documents are transformed automatically into XML behavior plans from which virtual character animations are synthesized via behavioral animation techniques.

The presented approach was successfully applied in several settings involving the animation of object manipulations performed by virtual humans. Several scenarios were realized by students as part of a university course such as different kitchen scenes (including the example described in this paper), a cockpit scene and an office desk scenario. The animated object manipulations comprise different grasp types as well as different types of constrained, unconstrained and touch actions of our action taxonomy. The open hierarchical 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 include the integration of head and foot movements or multiple (virtual) humans.

Acknowledgments

The research described in this paper was partially funded by the Deutsche Forschungsgemeinschaft in the “Virtual Workers” project. The authors thank the anonymous referees for their extensive and helpful comments.

References

[ACT05] Tolga Abaci, Jan Ciger, and Daniel Thalmann, *Planning with Smart Objects*, The 13th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision: WSCG, 2005, pp. 25–28.

[Arb02] Michael A. Arbib, *The Mirror System, Imitation, and the Evolution of Language*, Imitation in Animals and Artefacts (K. Dautenhahn and C. Nehaniv, eds.), MIT Press, 2002, ISBN 0-262-04203-7.

[BADV+08] Heni Ben Amor, Maik Deininger, Arnd Vitzthum, Bernhard Jung, and Guido Heumer, *Example-based Synthesis of Goal-directed Motion Trajectories for Virtual Humans*, 5. Workshop Virtuelle und Erweiterte Realität, GI-Fachgruppe VR/AR, 2008.

[BAHJV08] Heni Ben Amor, Guido Heumer, Bernhard Jung, and Arnd Vitzthum, *Grasp synthesis from low-dimensional probabilistic grasp models*, Computer Animation and Virtual Worlds **19** (2008), no. 3-4, 445–454, ISSN 1546-427X.

[BEL02] Norman I. Badler, Charles A. Erignac, and Ying Liu, *Virtual humans for validating maintenance procedures*, Commun. ACM **45** (2002), 56–63, ISSN 0001-0782.

[Cut89] Mark R. Cutkosky, *On Grasp Choice, Grasp Models and the Design of Hands for Manufacturing Tasks*, IEEE Transactions on Robotics and Automation **5** (1989), no. 3, 269–279, ISSN 1546-296X.

[DLB96] Brett Douville, Libby Levison, and Norman I. Badler, *Task-Level Object Grasping for Simulated Agents*, Presence **5** (1996), no. 4, 416–430, ISSN 1054-7460.

[GHJ+12] Frank Gommlich, Guido Heumer, Bernhard Jung, Matthias Lenk, and Arnd Vitzhum, *Enabling and Analyzing Natural Interaction with Functional Virtual Prototypes*, Modeling and Simulation in Engineering, Intech, 2012, pp. 261–279.

[Gle98] Michael Gleicher, *Retargetting motion to new characters*, SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques (New York, NY, USA), ACM, 1998, pp. 33–42, ISBN 0-89791-999-8.

[GMHP04] Keith Grochow, Steven L. Martin, Aaron Hertzmann, and Zoran Popović, *Style-based inverse kinematics*, ACM Transactions on Graphics **23** (2004), no. 3, 522–531, ISSN 0730-0301.

[Gui87] Yves Guiard, *Asymmetric division of labor in human skilled bimanual action: the kinematic chain as a model*, Journal of motor behavior **19** (1987), no. 4, 486–517, ISSN 0022-2895.

[HBAJ08] Guido Heumer, Heni Ben Amor, and Bernhard Jung, *Grasp recognition for uncalibrated data gloves: A machine learning approach*, Presence: Teleoperators and Virtual Environments **17** (2008), 121–142, ISSN 1054-7460.

[Heu10] Guido Heumer, *Simulation, Erfassung und Analyse direkter Objektmanipulationen in Virtuellen Umgebungen [Simulation, Recording and Analysis of Direct Object Manipulations in Virtual Environments]*, Ph.D. thesis, Faculty of Mathematics and Informatics, Technical University of Freiberg, Germany, 2010, nbn-resolving.de/urn:nbn:de:bsz:105-qucosa-70518.

[JAHW06] Bernhard Jung, Heni Ben Amor, Guido Heumer, and Matthias Weber, *From Motion Capture to Action Capture: A Review of Imitation Learning Techniques and their Application to VR-based Character Animation*, Proceedings VRST 2006 - 13th ACM Symp. on Virtual Reality Software and Technology, 2006, pp. 145–154, ISBN 1-59593-321-2.

[JBAHV11] Bernhard Jung, Heni Ben Amor, Guido Heumer, and Arnd Vitzthum, *Action Capture: A VR-Based Method for Character Animation*, Virtual Realities (Guido Brunnett, Sabine Coquillart, and Greg Welch, eds.), Springer Vienna, 2011, pp. 97–122, ISBN 9783211991770.

[Jun08] Yvonne A. Jung, *Animating and Rendering Virtual Humans: Extending X3D for Real Time Rendering and Animation of Virtual Characters*, GRAPP 2008: Proceedings of the Third International Conference on Computer Graphics Theory and Applications, 2008, pp. 387–394.

[KAAT03] Marcelo Kallmann, Amaury Aubel, Tolga Abaci, and Daniel Thalmann, *Planning Collision-Free Reaching Motions for Interactive Object Manipulation and Grasping*, Computer Graphics Forum **22** (2003), no. 3, 313–322, ISSN 1467-8659.

[Ken04] Adam Kendon, *Gesture: Visible Action as Utterance*, Cambridge Univ. Press, 2004, ISBN 9780521542937.

[KKN+02] James J. Kuffner, Satoshi Kagami, Koichi Nishiwaki, Masayuki Inaba, and Hirochika Inoue, *Dynamically-Stable Motion Planning for Humanoid Robots*, Autonomous Robots **12** (2002), 105–118, ISSN 0929-5593.

[KKW02] Alfred Kranstedt, Stefan Kopp, and Ipke Wachsmuth, *MURML: A Multimodfal Utterance Representation Markup Language for Conversational Agents*, Proceedings of the AAMAS Workshop on Embodied Conversational Agents - let's specify and evaluate them!, 2002.

[KL00] James J. Kuffner and Jean-Claude Latombe, *Interactive Manipulation Planning for Animated Characters*, PG '00: Proceedings of the 8th Pacific Conference on Computer Graphics and Applications (Washington, DC, USA), IEEE Computer Society, 2000, p. 417, ISBN 0-7695-0868-5.

[KMTA+02] Sumedha Kshirsagar, Nadia Magnenat-Thalmann, Guye-Vuillè Anthony, Daniel Thalmann, Kaveh Kamyab, and Ebrahim Mamdani, *Avatar Markup Language*, Proceedings of the workshop on Virtual environments 2002 (Aire-la-Ville, Switzerland, Switzerland), EGVE '02, Eurographics Association, 2002, pp. 169–177, ISBN 1-58113-535-1.

[KT99] Marcelo Kallmann and Daniel Thalmann, *Direct 3D Interaction with Smart Objects*, Proceedings ACM VRST 99, London, 1999, ISBN 1-58113-141-0.

[MM06] Minhua Ma and Paul McKevitt, *Virtual human animation in natural language visualisation*, Artificial Intelligence Review **25** (2006), no. 1-2, 37–53, ISSN 0269-2821.

[PAB+97] Jeffrey S. Pierce, Steve Audia, Tommy Burnette, Kevin Christiansen, Dennis Cosgrove, Matt Conway, Ken Hinckley, Kristen Monkaitis, James Patten, Joe Shochet, David Staack, Brian Stearns, Chris Sturgill, George Williams, and Randy Pausch, *Alice: Easy to Use Interactive 3D Graphics*, ACM Symposium on User Interface Software and Technology, 1997, pp. 77–78, ISBN 0-89791-881-9.

[PKS+02] Paul Piwek, Brigitte Krenn, Marc Schröder, Martine Grice, Stefan Baumann, and Hannes Pirker, *RRL: A Rich Representation Language for the Description of Agent Behaviour in NECA*, Proceedings of AAMAS 2002 workshop: Embodied conversational agents - let's specify and evaluate them!, 2002.

[RG91] Hans Rijpkema and Michael Girard, *Computer animation of knowledge-based human grasping*, SIGGRAPH Comput. Graph. **25** (1991), no. 4, 339–348, ISSN 0097-8930.

[Sch19] Georg Schlesinger, *Der Mechanische Aufbau der Künstlichen Glieder*, Ersatzglieder und Arbeitshilfen für Kriegsbeschädigte und Unfallverletzte (M. Borchardt et al., eds.), Springer-Verlag, Berlin, Germany, 1919, pp. 321–661.

[VCC+07] Hannes Vilhjalmsson, Nathan Cantelmo, Justine Cassell, Nicholas E. Chafai, Michael Kipp, Stefan Kopp, Maurizio Mancini, Stacy Marsella, Andrew N. Marshall, Catherine Pelachaud, Zsófia Ruttkay, Kristinn R. Thórisson, Herwin van Welbergen, and Rick J. van der Wert, *The Behavior Markup Language: Recent Developments and Challenges*, Proceedings 7th International Conference on Intelligent Virtual Agents, IVA-2007, Lecture Notes in Computer Science, vol. 4722, Springer, 2007, pp. 99–111, ISBN 978-3-540-74996-7.

[WF05] Ian H. Witten and Eibe Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd ed., Morgan Kaufmann, San Francisco, CA, USA, 2005, ISBN 0120884070.

[WHBAJ06] Matthias Weber, Guido Heumer, Heni Ben Amor, and Bernhard Jung, *An Animation System for Imitation of Object Grasping in Virtual Reality*, Proceedings of the 16th International Conference on Artificial Reality and Telexistence, LNCS, vol. 4282, Springer, 2006, pp. 65–76.

[Wor98] World Wide Web Consortium, *Synchronized Multimedia Integration Language (SMIL) 1.0 Specification*, 1998, http://www.w3.org/TR/REC-smil/ last visited March 30, 2011.

[YKH04] Katsu Yamane, James J. Kuffner, and Jessica K. Hodgins, *Synthesizing animations of human manipulation tasks*, ACM Trans. Graph. **23** (2004), no. 3, 532–539, ISSN 0730-0301.

Citation

Arnd Vitzthum, Heni Ben Amor, Guido Heumer, and Bernhard Jung, *XSAMPL3D: An Action Description Language for the Animation of Virtual Characters*, Journal of Virtual Reality and Broadcasting, 9(2012), no. 1, February 2012, urn:nbn:de:0009-6-32622, ISSN 1860-2037.