

# The generic description and management of interaction between autonomous agents and objects in an informed virtual environment

Marwan Badawi<sup>\*</sup>  
Bunraku Team  
IRISA  
Campus de Beaulieu  
35042 Rennes Cedex, France  
marwan.badawi@gmail.com

Stéphane Donikian  
Bunraku Team  
IRISA/INRIA  
Campus de Beaulieu  
35042 Rennes Cedex, France  
donikian@irisa.fr

## ABSTRACT

Autonomous agents cannot exist in an environment without interacting with the world surrounding them. In this paper we propose an informed environment based on synoptic objects which contain a synopsis of the interactions they can undergo. Through the use of interactive surfaces we manage to describe the surfaces of interest on the object itself and the space affected by the object during interaction. Then, by defining a set of seven basic actions, the objects can describe the interaction process through these actions to any agent implementing them. The description of the interaction process is done through complex actions which indicate the order in which the basic actions need to be accomplished and what to do depending on the result of the undertaken action.

## Keywords

Computer Animation, Informed Environment, Interactive Objects

## 1. INTRODUCTION

Creating believable virtual worlds populated by autonomous agents is a very demanding task which can be broken down into two main blocks: creating the virtual environment itself, and creating the autonomous agents populating it. However, to create a believable virtual world, the agents within it must be able to manipulate objects surrounding them and to interact with each other. Enabling agents to interact with their environment is the problem addressed by this article.

The proposed approach consists in generating real time interaction animation by storing generic information within the environment. To achieve this, the interactive object

---

<sup>\*</sup>Former PhD Student.

informs the agent of the interactions it provides, without taking the control on it. Furthermore, since the nature of an object does not change depending on the type of agent interacting with it, the interaction information needs to be generic and interpretable by any type of agent. This information thus is separated in two parts: the first part contains the information inherent to the object and is stored within the object itself, and the second part contains the information inherent to the agent and uses the object's information to determine the agent's behavior. The information stored within the objects is a general description of the interaction process, and we call such objects Synoptic Objects. They use Interactive Surfaces to describe the areas that take part in the interaction process and the space around the object affected by the interaction. Furthermore, an object describes the interactions it can undergo through a set of Basic Actions telling the agent what actions it needs to perform on the object when it wants to interact with it. These basic actions, coupled with their corresponding interactive surfaces, are used to describe the interaction process through Complex Actions.

## 2. RELATED WORKS

Autonomous virtual humans are able to perceive their environment, to communicate with others and to perform various activities in accordance with the nature of the environment and with their intentions. After gathering information from the environment through perception and after processing this information according to knowledge, plans and desires, the agent needs to act on the surrounding environment in order to achieve its established goal. There are studies that concern fine animation of grasping movements [3, 12] but it is not the purpose of this work. Our interest is how the agent interprets the information it gathers to accomplish complex interaction tasks within the environment, and manipulate objects and complex mechanisms. This type of approach is usually used in video games and is very well illustrated in *The Sims2* from Maxis Entertainment. This game mixes behavior and interaction information within the objects which advertise behavioral level services. Usually in video games all possible interactions with objects are recorded using motion capture and then replayed as is, completely unaltered. This gives extremely realistic animations, but this motion realism comes at the price of having the same animation and the same behavior repeated whenever an ac-

tion is performed, which tends to be unrealistic since real humans never perform exactly the same action twice. A hybrid approach is used by Badler et al. in the PAR system [1], where objects are integral parts in defining the action to be taken. The gestures accomplished by an agent during interaction are completely synthetic, and created by using the EMOTE model [2] which is an interpretation of the Laban Movement Analysis. A more generic approach is taken by Kallmann [8] with the Smart Object architecture. A Smart Object, which can hold a relatively complex action, will instruct the synthetic actor on the actions to do step by step. All the information necessary to interact with the object is contained in the object itself. A recent addition to the Smart Object architecture allows the running and mixing of simultaneous actions during interaction[?]. Information within the objects is given to an animation engine which can create and mix interaction animations in real time. Geometric information such as finger joints quaternion angles for a grasping action joints or a point in space for a look action are embedded directly within the object. So even though the animations themselves are generated dynamically, the interactions points and hotspots are themselves static and modeled offline.

Moving on to more Action oriented informed environments, a good example of such an environment containing interaction information is STEVE [11]. In STEVE, some basic information is stored inside the objects, to allow some simple manipulations, but the tasks it is able to perform are limited and very domain specific. It is used for training in virtual reality maintenance tasks aboard a ship and it is not generalizable. GIAT Virtual Training (GVT) is a similar environment which contains more complex manipulation information allowing inter-object as well as agent-object interactions [9]. GVT is scenario driven and allows the assembly and disassembly of very complex constructions but the fact that all interactions are fully scripted does not suit the haphazard quality of a real-life simulation. An even more generic environment is proposed by Jorissen et al. [7]. The animations of the object as well as the entire interaction paradigm are stored within the object itself and are accessible through triggers. This environment focuses solely on the description of the objects and does not take into account the animation requirements for interacting with autonomous agents.

### 3. CONCEPTUAL DEPENDENCY THEORY

Conceptual dependency is a theory established by R. Schank in 1969 [14] to make a computer understand text written in plain English. Conceptual dependency based itself on mapping text sentences to concepts understandable by a computer. Sentences are mapped in a conceptual structure representing an event, and composed of a fixed number of slots. Sometimes the slots cannot all be filled directly by the sentence, and a mechanism called slot filling is then used to infer the missing elements. Conceptual dependency allows unambiguous understanding of a sentence because it manages to map sentences with the same meaning, but using different words, to the same slot called primitive action.

At the heart of the theory is the conceptual structure called event. An event is always composed of an *actor* who triggers the event and is the one accomplishing the action, an *action*

performed by the actor, an *object* that is the recipient of the action or that the action is performed upon, and a *direction* describing the orientation of the action. The filling of this structure is not a simple task, since all the slots cannot always be directly deduced from the sentence itself. Some rules define the types of the slots, and what can be used to fill them [13]. The most important part of conceptual dependency is its use of primitive actions which represent atomic concepts. The primitive actions represent concepts rather than words, as sentences with completely different forms but with the same overall meaning need to have an identical representation. The eleven primitive actions are:

**atrans:** Transfer possession.

**mtrans:** Transfer information.

**mbuild:** Mental construction of new information from old information.

**ptrans:** Transfer of physical location.

**ingest:** Take something into the body.

**propel:** Apply physical force to an object.

**attend:** Focus sense organ to an object.

**speak:** Produce a sound or a noise.

**grasp:** Grasp an object.

**move:** Move a body part.

**expel:** Push something out of the body.

The last five actions, ATTEND, SPEAK, GRASP, MOVE and EXPEL are called instrumental actions. They are mostly used as a second representation called instrument that further describes the main conceptual representation. For example, the conceptual dependency representation of *John read a book*, would take John as the subject and MTRANS as an action since information is being transferred from the book to John. The instrument of this representation would thus be another conceptual dependency, having John as a subject, ATTEND as an action and eyes as the objects. Furthermore, the six main actions are further broken down into two categories: ATRANS, MTRANS and MBUILD represent mental processes, abstraction and reasoning whereas PTRANS, INGEST and PROPEL are physical actions representing movement and physical forces. Combined with the conceptual dependency representation of events, they enable unambiguous understanding of sentences, allow a simple classification of verbs into physical and mental processes, and determine which ones can be used as instruments of others.

Primitive actions can be used to describe the entire range of actions an autonomous agent can perform. This allows an easy description of the behavior of the agent by telling it which action to perform when it needs to perform it. Events can be used as templates for describing data. Not all the concepts of the primitive actions suit the needs of behavioral animation. Some are not of interest in this context, while others can be broken down into other primitive actions. Keeping in mind that our main goal is to produce low

level interaction animations without consideration for high level planning or reasoning, we did not take into account any of the mental actions. This brings us to our own set of seven primitive actions, which we call Basic Actions:

**transfer:** Change of the agent's physical location. Somewhat identical to PTRANS but the object and subject are always the same. It is used to tell the agent to go somewhere.

**move:** Move a body part of the agent. It has the same use as its conceptual dependency counterpart.

**grasp:** Tells the agent to grab an object. It simply connects the object to be handled with the agent's limb interacting with it, making the object follow the agent's movements.

**ingest:** Makes the agent take something into its body.

**expel:** Makes the agent expel something from its body.

**tell:** This action acts on a higher level than animation, since it allows to give information to the agent.

**attend:** Another high level action used in conjunction with tell to transmit specific sensory information.

Unlike conceptual dependency which uses events as a main building block, our approach uses basic actions as the principal structure. We find it more convenient to describe an agent's behavior as a series of actions to perform than as a chaining of events. Each action is composed of a structure similar to a conceptual dependency event and takes three parameters:

1. The actor who is accomplishing the action. It is mandatory and only accepts autonomous agents as slot fillers.
2. The object that is the recipient of the action. It is also mandatory and can be any type of object within the environment, another agent, or the actor itself.
3. The target of the action. This slot is optional since not all actions need a target. It can specify coordinates to reach for transfer or move, for example.

These are the actions that, in our opinion, are necessary to cover all the possibilities of animating autonomous agents interacting with their environment. The first set of basic actions we had proposed was somewhat different [left blank], and we have had many additions and modifications since then. But now, we are confident that these actions respond to the needs of creating most animations for agent interaction.

#### 4. INTERACTIVE SURFACES

When wishing to interact with an object, the first thing humans do is determine its functionality through visual clues which have been called affordances [5] or knowledge in the world [10]. By analyzing visual data, humans can determine the points of interest in an object and infer spatial positioning data relative to the interaction process. But

when creating virtual humans, simulating such a process time consuming and extremely complex. This is why we use an informed environment, already containing this type of information: each object within the environment advertises its affordances through interactive surfaces that directly provide the necessary spatial information to the agent. Interactive surfaces describe what the objects offer in terms of interaction information, or in other terms, what they afford. There are two kinds of interactive surfaces:

**Interaction surfaces:** these surfaces are generally a part of the object's geometry. They define the areas of the object that are used during interaction. They describe the parts of the object in contact with the agent during interaction.

**Influence surfaces:** these surfaces usually do not belong to the object's geometry. They define the areas surrounding the object where the agent has to position itself in order to interact with the object. They describe the space affected by the object during interaction.

Throughout the rest of this paper, the term interactive surface (IS) will be used to refer indifferently to both types of interaction and influence surfaces. ISs do not only describe the affordances of interactive objects within the environment, but also the interaction capabilities of the autonomous agents. For example, ISs on the palm of the agent's hands would allow it to use its hands for grabbing objects, while an IS on its backside would allow it to sit on a chair, etc. In order to better describe affordances, an IS is always associated to a basic action. This way the agent knows where to GRASP an object or where to TRANSFER itself. Interaction surfaces act as hot spots on the object itself whenever an agent is interacting with the object. They designate areas of the object that are pertinent to the interaction process. Figure 1, shows a simple interaction surface on a coffee mug associated with the move action. As can be seen in the figure, the interaction surface is composed of distinct geometric pieces: one on the handle and one on the mug's body. Even though they are geometrically separate, the areas marked in red are considered to be a single interactive surface since they describe the same affordance and correspond to the same basic action.

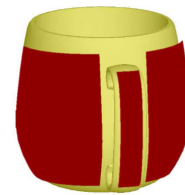
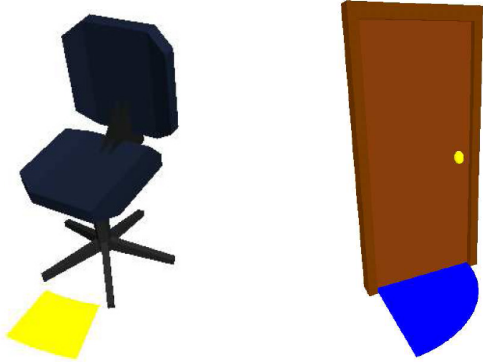


Figure 1: A mug with a red interaction surface.

Interaction surfaces do not always correspond to the object's geometry. Objects that work by sensing proximity, like automatic doors and electronic eyes, have interaction surfaces that do not even correspond to a visual representation. Influence surfaces are the second type of IS. They describe areas surrounding the object and act as indicators of the area influenced by the interactions the object affords.

These surfaces are mainly used to correctly position the agent during the interaction process. In short, they describe the space affected during interaction. Influence surfaces can have a positive influence by describing an area where the agent needs to be present to interact with the object. Figure 2(a) shows the positive influence surface of a chair. It indicates the area in which the agent must be situated in order to sit on the chair. Influence surfaces can also have a negative influence describing the areas an agent must avoid being in when interacting with the object. Figure 2(b) shows the negative influence surface of a door. It indicates the area swept by the door when it is being opened thus indicating an area the agent should avoid during interaction.



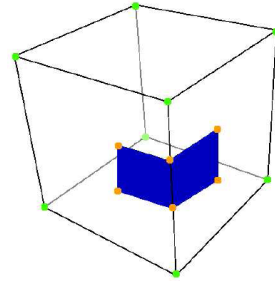
(a) A positive influence surface for a chair, in yellow. It shows where the agent needs to be placed before sitting.

(b) A negative influence surface for a door, in blue. It shows the area swept by the door when it is opened.

**Figure 2: Influence surfaces.**

An interesting property of the door's influence surface in 2(b) is that it can be also positive. If an agent is opening the door for someone else, he would have to open the door by both avoiding the door, and staying out of the way of the other agent. This could be achieved by considering the influence surface as positive, forcing the agent to stay within it when opening the door. Even though the majority of influence surfaces describe the surroundings of an object, they can sometimes be part of the object's geometry itself. Objects that work by moving parts of themselves, like conveyor belts or elevators, have influence surfaces corresponding to their moving parts.

The information given by the surface itself is largely insufficient to calculate a precise trajectory for an arm, for example, or for creating any kind of interaction animation. But this lack of precision is exactly what we are seeking, since it does not define a very deterministic way of interacting with objects. This non-deterministic element in creating interactions between agents and objects is one of our main goals since it adds a form of realism by eliminating the repetitiveness found in other systems. The same agent, interacting with the same object, will never execute the same animation twice because it will never place itself at the exact same position or put its hand on the exact same spot every time.



**Figure 3: A cube with an interaction surface.**

The idea behind the creation of interaction surfaces is pretty straightforward: load a 3D object into the synoptic object editor then select the interactive area on it. The transition from the 2D selection to the 3D projection on the object itself needs to resolve two problems: first, the 2D surface needs to be projected within the object space. Second, the 3D surface has to have its own representation, independently of the elements composing the object it is projected on. 3D representations of an object consist in setting specific points in space, called vertices, then connecting them with lines creating polygons outlining the object's shape. Figure 3 shows the green vertices that represent the cube and the orange vertices that represent the blue interaction surface. The vertices of the interaction surface do not correspond to the vertices of the underlying cube. To solve these problems we used OpenGL for manipulating the 3D data, in particular the OpenGL feedback buffer. The process used by OpenGL for drawing 3D environments and then projecting them on the screen in 2D is called rasterization. When only the visible part of an object is rasterized, the non-visible parts of the object are removed through a clipping process. When an object is clipped, some of the polygons composing it are clipped too. When these clipped polygons are rasterized, they are represented using new points on the screen that do not correspond to the original vertices composing them. These new points, along with the polygons rasterized to the screen, are stored in the OpenGL feedback buffer. Furthermore, along with its 2D coordinates on the screen, each rasterized point possesses other types of information and, in particular, normalized depth information. This information allows to calculate the depth in the 3D scene corresponding to the point visible on the 2D screen. By gathering these two types of information from the feedback buffer, it is possible to recover the new points drawn on the screen then project them back into 3D space to create an interaction surface. By re-projecting the contents of the feedback buffer onto the object, we obtain surfaces that perfectly correspond to the object. However, this perfect correspondence is not always a good thing, especially when the object has a rugged and irregular surface. Such an object will have a similarly irregular interaction surface which need to be smoothed out and simplified. A humanoid agent wishing to place the palm of its hand on such an IS could place it at a position that would make its hand penetrate the object. By simplifying the interaction surface, it is possible to avoid such errors. But the simplification cannot be done arbitrarily, and must be made to correspond to the underlying object as best as possible. To solve this problem, we use a 3D convex hull algorithm provided by CGAL, the Computational Geometry

## 5. RETRIEVING INTERACTION INFORMATION

For every interaction, there are always two parts involved: the one doing the interaction and the one receiving the interaction. So, for each interaction, two ISs are concerned: one on the agent and one on the object. An agent wanting to grab the handset of a phone would have to move its arm so that the IS on the palm of the agent's hand comes into contact with the IS on the handset. In order to accomplish that, three pieces of information are needed: a point of contact on the agent's IS, a point of contact and a contact normal on the object's IS. To place the agent's hand at the correct position, it would simply imply touching the point on the agent's hand to the point on the handset and insure the correct orientation of the hand by aligning it with the contact normal. Thus, it is necessary to predict a collision between the two surfaces, then make it happen by feeding the information to the animation engine.

A very important factor to keep in mind is the shape of the ISs, as it is very rarely convex, and may not even describe a closed volume. This lack of constraints on the shape of the ISs gives us the freedom to choose an IS by concentrating on the functionality instead of worrying about mathematical considerations relating to the collision detection process. But this freedom comes with a price, since predicting and detecting collisions between non convex objects is very tedious. To that effect, we use a collision detection engine called SWIFT++ [4]. Besides the fact that it allows collision prediction, SWIFT++ can be used to manipulate non convex objects and is, to our knowledge, the only collision detection engine capable of doing so. So, when the agent needs to interact with an object, it tries to predict a collision between the IS of interest on the object, and the IS it wants to use to interact with that object. In our example, it would be the IS on the handset and the IS on the agent's palm. The ISs are fed into the SWIFT++ engine and a collision prediction is requested. A property of SWIFT++ is that it does not return a single collision point, but a set of potential candidates. These points are defined by their position and their normal, indicating the direction of the collision. A set of contact points and normals for each IS are returned. We assume that the IS on the agent is accurately defined and that any point on it is acceptable for use. Then the collision information on the object's IS is sorted according to two criteria:

1. Point proximity: This criterion favors the points closest to the point chosen on the agent's IS. It ensures minimal displacement of the agent also that he interacts with the part of the object closest to it.
2. Normal proximity: This criterion favors the normals whose orientation is closest to that of the point chosen on the agent's IS. It ensures minimal modification of the agent's posture and minimizes the movement adaptation.

The selection can be done according to one of these criteria, or a mix of both. Once the most adequate data for the

predicted collision is chosen, the agent has a precise point on the object's surface with which to act and a normal at that point giving him a precise angle of approach.

## 6. BASIC AND COMPLEX ACTIONS

In conjunction with ISs, basic actions manage to describe where interaction should occur, and how. The basic actions previously defined, do not have the same degree of complexity, from an animation point of view. Some, like MOVE, are very complex and take into account many factors: what part of the body is being MOVE-ed, what the target is, etc. Others, like TELL, are just specifying the sensor and the information to TELL. However, they share the same structure, specifying an actor performing the action and an object receiving the action. Some of the basic actions also need a third information, a target which can contain different types of information, such as spatial coordinates to transfer to, or the information to TELL.

TRANSFER is used by the actor to transfer itself to a new physical location. TRANSFER mostly has influence surfaces as targets because they describe the positive or negative influence of an object, the placement of the agent near and around the object. It is the first action to accomplish before any interaction, gathering information from the object's influence surface then placing the agent according to that information. Since we are working on the level of interaction description, the transfer action does not contain any high level information for path planning and collision avoidance. When the agent is given a target to transfer to, it calculates a speed vector  $\vec{v}$  going from the agent's position toward the target. The speed vector is broken down into two components on the front-back and the left-right axes which are then used to determine the importance each direction must have in the final resulting animation. At the same time, a desired direction angle  $\hat{d}$  is calculated, between the agent's current direction, and the direction it would have when facing the target. The speed vector and the direction angle are calculated at each simulation step, and the mixing of the locomotion animations modified accordingly. Finally, when the target is reached, the last step is modified using footprint constraints to place the feet at the precise position of the target before stopping the locomotion animations.

MOVE is used to move an agent's body part to a specific location. The most common limbs that need to be MOVE-d are, by far, the arms. Most interactions done by humans need the arms and hands which explains why special attention has been brought to these limbs. But MOVE is not restricted to the arms, it can, for example, be used to lower the head to avoid a hanging object, place the backside correctly on a chair, etc. It is a very versatile action and its implementation keeps growing with the interaction needs of new synoptic objects. For the sake of clarity, we will only describe the MOVE action for the arms, the other limbs being controlled in the same way. MOVE is used in two stages: choosing the best suited original motion and adapting it to the desired location. Like TRANSFER, MOVE uses multiple motion animations and chooses the one that is best suited for the task at hand. The arm needs to be MOVE-d to the location given by the object's IS. To that effect, a spacetime constraint is placed on the palm of the hand so that it will be placed at the desired location when the animation ends.

The original animation will thus be modified smoothly and the result correspond to the interaction underway. If the agent is using a tool to interact with the object, the IS of the tool is used to determine the contact point on the tool itself. This position is then used to calculate an offset between the hand holding the tool and the point on the tool's IS. This offset is used to displace the hand's spacetime constraint to accordingly place the tool at the desired position instead of the hand itself. The final position of the constraint is set on the target synoptic object's IS. At the end of the motion, the reference motion will have been gradually modified so that the final position of the palm corresponds to the position of the predicted collision on the object. When the collision prediction is performed, the normal at the collision point on the object is also recovered. Throughout the execution of the motion, the rotation of the wrist is modified so that the palm is perfectly perpendicular to the normal once it reaches its target.

GRASP, like the next four basic actions, is independent of the agent's morphology. Its first purpose is to inform the object being interacted with to MOVE when the limb that is GRASP-ing it MOVE-s. Its second purpose is to tell the limb that GRASP-s the object, to stop using its own IS and start using the GRASP-ed object's IS instead. When an object is GRASP-ed, it creates an input and connects it to the corresponding limb's output. The position given by the limb's output when GRASP is used, corresponds to a point on the object's IS. Since the object can be grabbed from anywhere on its IS, when the limb is MOVE-d, it is that point that needs to follow the limb's movement. To that effect, the GRASP-ed object calculates an offset between its current position, and the position of the limb it is attached to. At each calculation step, the object reads its connected input and sets its position accordingly by adding the calculated offset to it, thus following the movements described by the agent's output. GRASP is closely linked to move. For example, if the agent GRASP-s a glass, it will need to use an IS on the bottom of the glass if it wants to MOVE it to a table and set it down. On the other hand, it will use an IS on the rim of the glass if it wants to MOVE the glass to its lips and drink from it.

INGEST and EXPEL are not very complicated actions. The animation aspect of these actions is very simple: when an object is INGEST-ed, it is simply removed from the environment and when it is EXPEL-ed, it is added to the environment. Each synoptic object contains a set of three integers indicating its size, capacity and available space. An object can only INGEST objects whose size is smaller or equal to the containers available space which cannot exceed its capacity. This is a simplified container model which does not take into account the many different physical factors necessary for a more complete container model. Even so, this model is sufficient for our actual needs and can easily be extended to create more complicated relationships between objects. The use of EXPEL is pretty straightforward. It is initialized with a container and an object to be EXPEL-ed. It updates the available space of the container and sends a *Resume* event to the controller to reactivate the previously ingest-ed object. The TELL and ATTEND basic actions do not have an effect on the animation of the agent. They are used to gather information from the environment, and operate at a slight-

ly higher cognitive level than the rest of the basic actions. Whenever an agent needs to recover information from an object, it uses ATTEND to send a valued event to the desired object. The value of the event is the sensor being used to ATTEND to the object. Each object has different sets of data associated to different types of sensors. The receiver looks up the information relevant to the used sensor, packs it inside a valued event with a tell identifier, and sends it back to the agent paying attention to it.

## 7. RESULTS AND EXAMPLES

The first example shows an agent using a bottle to fill a glass. The synoptic bottle and glass can be seen in figure 4(a). The yellow ISs on both objects are used by move before grasp-ing the object. The red ISs are used by move to constrain the bottle's neck to the glass's rim, to ensure the correct position of the bottle on the glass when pouring. The first part of the simulation consist in move-ing the arms to the correct positions then grasp-ing the glass and bottle. The actions can be done in parallel or in sequence since they use two different limbs and are thus not in conflict.



(a) The yellow and red ISs.

(b) The resulting action without and with the constraint.

**Figure 4: An agent pouring water from a bottle into a glass.**

The second example consists in an agent opening a door. Figure 5 describes the behavior of a synoptic door. It is composed of several FSMs that represent the complex actions used for the door, and the states the door can be in. The Door State FSM is used to tell the agent if the door is open or closed, and the actions the agent can apply to the door. If the door is in the open state, the agent can perform the Close Door action. If the door is in the closed state, the agent can perform the Open Door action. The door can also be in an undetermined state if an agent decided to stop interacting with it before it was fully open or closed. In this case, the agent can either perform the Open Door or Close Door action.

The Close Door and Open Door FSMs are the two complex actions that can be undertaken by the door. When normally undertaken, the actions transit through the states represented in black. For the Open Door FSM, it would be: (1) TRANSFER agent to the door's threshold; (2) MOVE the agent's right arm to the doorknob; (3) GRASP the doorknob with the right hand; (4) MOVE the right hand to turn the doorknob; (5) MOVE the right arm to open the door; (6) GRASP the doorknob again to let it go.

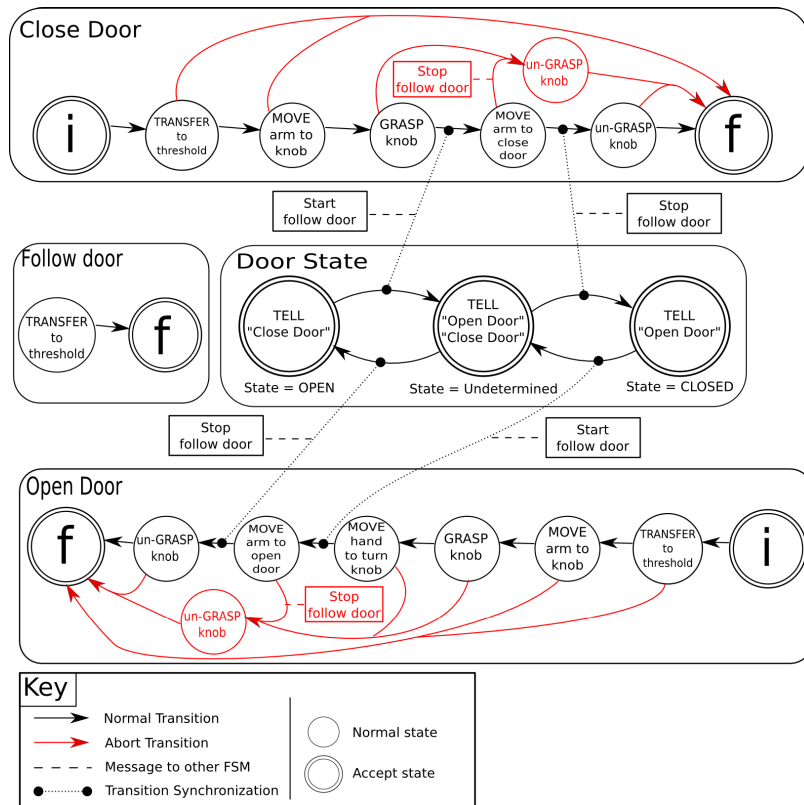


Figure 5: The complete behavior of a synoptic door.

While performing this complex action, the FSM is synchronized with two other FSMs. When step 5 begins, the door starts opening. Since it is no longer closed and it is not yet fully open, a message is sent to the Door State FSM which transits from the closed state to the undetermined state. At the same time, the Follow Door FSM is started. Since the door is moving now, the agent will need to follow the door with its movement. This is done by making the agent constantly transfer itself inside the door's IS representing the threshold. Since the IS is linked to the door's geometry, it will move with it and thus the agent will follow the threshold's movements. When step 5 is completed, the door is completely open. When the Open Door FSM transits from step 5 to step 6, the Door State FSM is updated accordingly and the Follow Door FSM is stopped. The door is now open. When the door is being opened or closed, it is the door itself that is moving, and the agent's hand is simply constrained to the knob's position. That way it is possible to modify the door's way of opening and closing without affecting either the complex action, or any of the associated ISs. The door can be made to open by pulling, pushing or sliding and the agent will still be able to perform the Open Door and Close Door actions, without any modification to the action's description. These three modes of opening the door are illustrated in figure 6.

## 8. CONCLUSION AND PERSPECTIVES

Throughout this paper we have presented our work on interactive objects within virtual environments. We have proposed the notion of synoptic objects which contain informa-

tion describing the interactions they propose, independently of the agent using them. The interaction description consists in complex actions, whose building blocks include interactive surfaces and basic actions. The agent wishing to interact with the objects, interprets the basic actions in its own way, thus interacting with the object in the manner most adapted to the agent itself. The seven basic actions create a small simple and invariant group of atomic actions diverse enough to describe all the needs for creating interaction animations. Through their associated resources, complex actions allow an agent to know which actions it can perform in parallel, or what resources to obtain in order to accomplish a certain action, like getting a key for unlocking a door. Interactive surfaces allow the agent to create dynamic interaction data on the fly, and thus adapt the interaction animation to the situations the agent and the object are in. No animation sequence is predetermined and the movements of the agent are entirely dynamic and created on-line to fit the current situation.

Although our approach offers many contributions to the field of behavioral animation, it also suffers from some drawbacks. The main drawback of our approach is the definition of interaction surfaces. Selecting a surface on a very complex object can be very tedious if the user wants to be sure that none of the selected surfaces will lead to a bad placement of the limbs. This problem however, can be solved by sacrificing precision and replacing the problematic object by its convex hull. Our system is also unable to coordinate the efforts of two agents acting together on the same object.

## 9. REFERENCES

- [1] T. Abaci. *Object manipulation and grasping for virtual humans*. PhD thesis, École Polytechnique Fédérale de Lausanne, March 2006.
- [2] N. Badler, R. Bindiganavale, J. Allbeck, W. Schuler, L. Zhao, S. Lee, H. Shin, and M. Palmer. Parameterized action representation and natural language instructions for dynamic behavior modification of embodied agents. In *AAAI Spring Symposium*, 2000.
- [3] M. Costa, L. Zhao, D. Chi, and N. Badler. The EMOTE model for effort and shape. In *SIGGRAPH*, 2000.
- [4] B. Douville, L. Levison, and N. Badler. Task-level object grasping for simulated agents. *Presence*, 5(4):416–430, 1996.
- [5] S. Ehmann and M. Lin. Accurate and fast proximity queries between polyhedra using surface decomposition. In *Eurographics*, 2001.
- [6] J. Gibson. *The Ecological Approach to Visual Perception*, chapter 8: The Theory of Affordances, pages 127–143. Lawrence Erlbaum Associates, 1979.
- [7] S. Hert and S. Schirra. 3d convex hulls. In C. E. Board, editor, *CGAL-3.2 User and Reference Manual*. 2006.
- [8] P. Jorissen and W. Lamotte. A framework supporting general object interactions for dynamic virtual worlds. In *4th International Symposium on Smart Graphics*, pages 154–158, May 2004.
- [9] M. Kallmann. *Object Interaction in Real-Time Virtual Environments*. PhD thesis, EPFL, Switzerland, 2001.
- [10] N. Mollet and B. Arnaldi. Storytelling in virtual reality for training. In *Edutainment*, 2006.
- [11] D. Norman. *The Design Of Everyday Things*, chapter 3: Knowledge in the head and in the world, pages 54–80. Basic Books, 1988.
- [12] J. Rickel and W. L. Johnson. Animated agents for procedural training in virtual reality: Perception, cognition, and motor control. *Applied Artificial Intelligence*, 13:343–382, 1999.
- [13] R. M. Sanso and D. Thalmann. A hand control and automatic grasping system for synthetic actors. In *Eurographics*, 1994.
- [14] R. Schank and C. Riesbeck. *Inside computer understanding: five programs plus minitaur*. The Artificial Intelligence Series. Lawrence Erlbaum Associates, 1981.
- [15] R. Schank and L. Tesler. A conceptual dependency parser for natural language. In *Proceedings of the 1969 conference on Computational linguistics*, pages 1–32, Sång-Säby, Sweden, 1969.



(a) An agent pulling to open a door.



(b) An agent pushing to open a door.



(c) An agent sliding to open a door.

**Figure 6: An agent opening a door in three different manners.**