Task-level Object Grasping for Simulated Agents

Brett Douville
University of Pennsylvania

Norman I. Badler
University of Pennsylvania, badler@seas.upenn.edu

Libby Levison
University of Pennsylvania

Task-level Object Grasping for Simulated Agents

Abstract
Simulating a human figure performing a manual task requires that the agent interact with objects in the environment in a realistic manner. Graphic or programming interfaces to control human figure animation, however, do not allow the animator to instruct the system with concise "high-level" commands. Instructions coming from a high-level planner cannot be directly given to a synthetic agent because they do not specify such details as which end-effector to use or where on the object to grasp. Because current animation systems require joint angle displacement descriptions of motion - even for motions that incorporate upwards of 15 joints - an efficient connection between high-level specifications and low-level hand joint motion is required. In this paper we describe a system that directs task-level, general-purpose, object grasping for a simulated human agent. The Object-Specific Reasoner (OSR) is a reasoning module that uses knowledge of the object of the underspecified action to generate values for missing parameters. The Grasp Behavior manages simultaneous motions of the joints in the hand, wrist, and arm, and provides a programmer with a high-level description of the desired action. When composed hierarchically, the OSR and the Grasp behavior interpret task-level commands and direct specific motions to the animation system. These modules are implemented as part of the Jack system at the University of Pennsylvania.

Task-level Object Grasping for Simulated Agents

Brett Douville & Libby Levison & Norman I. Badler

Center for Human Modeling and Simulation

Department of Computer and Information Science

University of Pennsylvania,

Philadelphia, PA 19104-6389

{brett@linc & libby@linc & badler@central}.cis.upenn.edu

Keywords: Grasping, Object Manipulation, Animation,

Animated Agents, Task-Level Control

Abstract

Simulating a human figure performing a task requires that the agent interact with objects in the environment in a realistic manner. In this paper we describe a system which directs task-level, general-purpose, object grasping for a simulated human agent.

The Object Specific Reasoner (OSR) generates parameters for underspecified task-level instructions such as `(pickup jack hammer)`. The Grasp behavior manages simultaneous motions of the joints in the hand, wrist and arm. When composed hierarchically, the OSR and the Grasp behavior interpret task-level commands to the animation system. These modules are implemented as part of the *Jack* project at the University of Pennsylvania.

1 Introduction

Our research is concerned with building a general-purpose system to animate a simulated human character manipulating objects. This paper focuses specifically on grasping tasks. Grasping is one of the most complex of the agent-object manipulation tasks. The human hand has 15 joints and 20 degrees of freedom. To generate a realistic grasp of an object, each digit of the agent's hand must close down around the object simultaneously. The wrist must be compliant as the palm moves to fit to the object. For an animator to enumerate the motions required by each involved joint is difficult and tedious. In circumstances where an animator is not used, for example, in virtual reality manipulations of real-tie task simulation, a program must create the appropriate agent/object grasp interactions.

We are building a task-level interface which allows the animator to instruct the system with intuitive, concise commands. This interface generates the individual joint motions required by the behavior and integrates two special-purpose components: an intermediate reasoning system (the Object Specific Reasoner) and an agent motor behavior which simulates grasping.

1.1 Jack

The research described here uses software developed in the Center for Human Modeling and Simulation (HMS) at the University of Pennsylvania. The *Jack*® modeling system runs on Silicon Graphics workstations and provides 3D-modeling capabilities, as well as extensive human factors and analysis tools [BPW93]. *Jack* provides a simulation system and *behaviors* for any agent [BB93]. Behaviors manage and schedule sets of joint motions to execute simultaneously. Grasping has been implemented as one of the basic *Jack* motor behaviors.

Becket [BB93] provides an integrated behavioral architecture tightly bound with the *Jack* virtual environment, as illustrated in Figure 1. The unshaded modules in Figure 1 illustrate the components (and their integration) described in this paper. As is shown, adding these two modules will, hopefully, span the gap between the high-level planner and the *Jack* virtual environment.

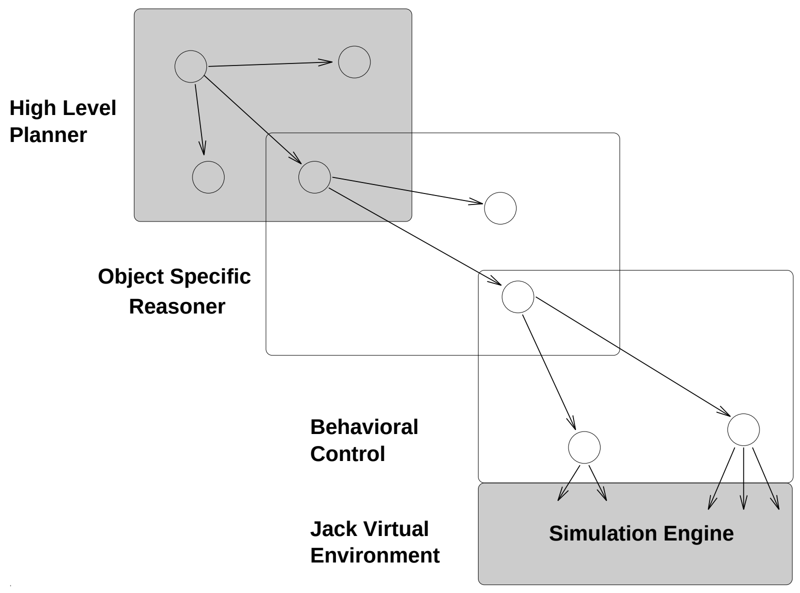

Figure 1: Placement of the OSR and Behavioral Simulator in system.

1.2 Parallel Automata: PaT-Nets

Parallel Transition Networks (*PaT-Nets*) are control structures which execute within *Jack*'s simulation system. A full discussion of the PaT-Net is not within the scope of this paper; for a more detailed description, see [CPB$^+$94, MGR95a, MGR95b, Bec94]). PaT-Nets are essentially finite state automata which execute in parallel, consisting of nodes connected by directed arcs called *transitions*. An *action* is associated with each node; this action consists of arbitrary LISP code which can directly (for example, by adjusting a joint angle) or indirectly (for example, by constraining the arm to move to a certain location) affect the simulation. Each transition has an associated *condition*, an arbitrary piece of LISP code which evaluates to true or false. On each simulation step, the action associated with the current node is executed; the first transition with a true condition is taken. The simulation proceeds at the next clock tick by evaluating the action of the node to which the PaT-Net transitioned, and so forth. Although evaluated code can perform arbitrary computations, invocations of *Jack* motor behaviors are of particular interest.

Several PaT-Nets can exist in a single simulation; indeed, several instances of the same PaT-Net can execute simultaneously, as in the grasping PaT-Nets described below.

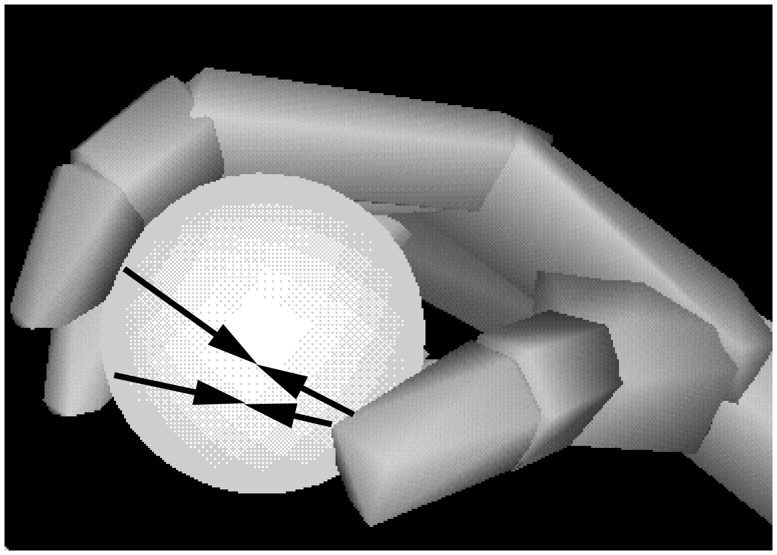

Figure 2: An Example Opposition Space: The thumb and first two fingers exert equal and opposite forces in order to grasp the ball stably.

2 Simulated Human Grasping

2.1 An Overview of Human Prehension

The opposition of the thumb and fingers is the most important ability of the human hand, allowing for the highly adaptive manipulation we call *grasping* [Nap93]. Our most basic interactions with the world involve the manipulation of objects with our hands; even the infant quickly develops schemas for basic grasping [Pia53]. Any simulation of task-level human abilities must account for this most basic of human skills.

Recent work in robotics and cognitive science describes stable prehensile grasps (*i.e.* grasps adapted for manipulation) in terms of *opposition spaces* [IBA86, MI94]: grasps in which an *opposition vector* exists between surfaces of the hand. Figure 2 shows a three-fingered grasp between the pads of the fingers, with the opposition vectors overlaid. An opposition vector describes the relationship between two or more *virtual fingers*, which exert forces sufficient to control the target object along the line of opposition. Once a mapping from virtual fingers to real fingers has been determined, a stable grasp has been achieved.

There are three basic types of oppositions used in human prehension:

- PAD opposition: the opposition vector runs between hand surfaces in a direction parallel to the palm (see Figure 2).
- PALM opposition: the opposition vector runs between hand surfaces in a direction generally perpendicular to the palm.
- SIDE opposition: the opposition vector runs between hand surfaces in a direction generally transverse to the palm.

In human prehension, two steps follow after the selection of a particular grasp for a specific task. In the first step, the hand preshapes while the palm orients to the object. Preshaping involves opening the hand sufficiently to span the object [MML90]. Palm orientation, as described in [Ibe87a, AIL85], is a "ballpark" process in which the hand nears the object but has no specific target location relative to the object. The second step involves a tactilely driven finger closure in which the hand attempts to close into the selected grasp posture while using sensory information from the hand surfaces to adapt the grasp to the target geometry.

Finally, studies of grasping (for a survey, see [MI94]) have suggested that prehension proceeds serially, with individual components of the grasp occuring in parallel. For example, while preshaping the hand, the palm is oriented and transported towards the target. The fingers close in parallel with the thumb and further refinements to the position of the palm. A grasping system should therefore incorporate both the serial and parallel aspects of human prehension.

2.2 Previous Approaches to Simulated Grasping

There is some previous work in the graphic simulation of automatic human prehension, notably [RG91, MTN88, MTT91] and [KKKL94]. Rijpkema and Girard present an inverse kinematics approach to control of the fingers and thumb. This computationally expensive approach denies the close connection between sensory input and the resultant grasp. Furthermore, they concentrate primarily on pad grasps, neglecting the palm opposition usually seen in power grasps.

Koga, *et al* also address the issue of grasping and the more general problem of multi-arm manipulation [KKKL94]. Their grasping approach requires significant overhead: objects to be manipulated must be tagged with specific target sites for each finger and for each different grasp. Grasping is then treated as an inverse kinematics problem where the goal sites have been determined in advance. Their work also addresses multi-arm manipulation and a special-case form of path-planning, whereas our work seeks to describe task-level general instruction commands particularly related to grasping.

Magnenat-Thalmann [MTN88, MTT91] provides an interesting account of human body deformations and the interaction between the forces of a gripping hand and a deformable object, but does not address the issue of grasping, only the animation of realistic deformation.

The robotics literature contains a significant amount of work in automated grasping. Iberall [Ibe87b] proposes a two-level architecture in which:

1. An object and task representation are mapped into a grasp-oriented task description.
2. The task description is mapped onto a grasping schema, whether an opposition space, tactilely driven control, or analytical framework.

An approach by Stansfield [Sta90] describes a two-level architecture for object grasping, but this can be distinguished from the two-level architecture above, in that this is really a two-*phase* architecture. In the first phase, the agent reaches for the object; in the second, the hand is constricted around the object. While Stansfield uses some symbolic knowledge, it is not clear that the approach is sufficiently general for human grasping.

Cutkosky and Howe [CH90] present a hierarchy of manufacturing grasps which subdivides Napier's original distinction of *power* and *precision* grasps [Nap93] into 16 distinct grasps, which are reflected in Figure 4 below. This hierarchy represents grasps for a number of object geometries (from spheres to cylinders to disks and cones) and task goals (from lifting to turning to exerting torque).

Some recent efforts have focused on heuristics for selecting grasps based on object geometry and task information (see, for example, [BLTK93], an extension of [TBK87]). Bekey focuses on selecting both grasp types and grasp locations based on object representations, task specifications, and object geometry, using both heuristics and a certain degree of table look-up. Tomovic's earlier paper addresses geometric models for classes of objects and the appropriate finger closing algorithms for these. Other examples of knowledge-based grasping techniques include [Sta90], [IJLZ88], and [Ibe86].

2.3 Grasping in *Jack*

As discussed in above, the second phase of human prehension constitutes the low-level "motor control" of grasping, namely, the tactilely driven closure of the hand. During this phase, the human hand (the fingers, thumb, and palm) relays signals about contact with the object geometry back to the central nervous system, which sends back commands on how to tailor the grasp to the object geometry. This process continues until the hand grasps the object stably.

In *Jack*, we simulate the sense of touch by using collision detection. Collision detection and an input parameter describing the desired opposition drive the closure of the fingers. The sections below describe this approach in more detail.

2.3.1 Simulating Tactile Sensation

Human prehension proceeds from tactile sensations delivered from the surfaces of the hand to the central nervous system. This allows for the perception of object properties that might not be immediately obtainable by visual inspection (for example, object texture is often not visually apparent). Furthermore, tactile sensing enables stable grasping by virtue of the direct accessibility of grasping forces along the opposition vectors; in short, the fingers can feel the forces they are exerting. Tactile sensation enables the perception of the interacting forces between hand and object which lead to a stable grasp; usually, the visual system cannot perceive these oppositions.

In *Jack*'s simulated grasping, collisions between the fingers and the target object trigger transitions in the PaT-Nets which simulate the hand closing process. For example, in a power grasp, where the palm is generally brought into contact with the target before the fingers close, a collision between the proximal digit of the finger and the target object would trigger a transition into a state in which the two distal segments of that finger are closing while the proximal segment remains in place. Power grasps require a high degree of contact between the surfaces of the fingers and palm with the object to be grasped; therefore, a collision between the proximal digit and the object would be likely to be maintained as part of a stable grasp.

The transitions taken in these PaT-Nets depend on the desired opposition. For example, the above collision between the proximal digit of the finger with the target object would trigger an entirely different transition if the goal opposition were a pad opposition; specifically, it would pull that finger back a little bit to ensure a pad opposition. Thus, the same control structures generate different types of grasps based on the desired opposition type.

2.3.2 Parallel Execution

In order to achieve realistic looking grasps, the fingers need to close in parallel. Building a single controller for the entire hand, however, requires a complete description of the possible interactions of the fingers, thumb, and target object. Since *Jack*'s hand has 20 degrees of freedom, determining the number of possible interactions is a daunting task. Alternatively, a great deal of apparent complexity can be achieved using simple PaT-Nets which interact to achieve a stable grasp.

Figure 3 shows a power-grasped hammer (a palm opposition grasp along the axis of the hammer handle). Each finger and the thumb acted in parallel to achieve this grasp. Collisions between any finger and the object (or other fingers) was the responsibility of the PaT-Net controlling that finger. Five PaT-Nets executing simultaneously and independently produced the grasp shown in Figure 3.

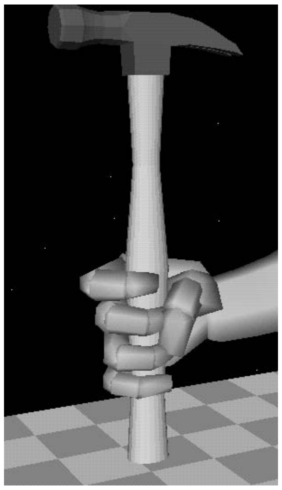

Figure 3: Power Grasping A Hammer

2.3.3 Implemented Grasps

The fifteen grasps shown in Figure 4 are based on Cutkosky and Howe's sixteen grasps in [CH90]. (The sixteenth, the hook or platform push, is left out because it is not prehensile.)

All fifteen grasps are accomplished by three simple PaT-Nets (two of which may be multiply instantiated during a particular grasp) in order to control the four fingers. The first of these controls the closing of the finger. Each of *Jack*'s fingers is similar in structure, consisting of two degrees of freedom at the base, and one each at the medial and distal joints, although the geometry and scaling may be somewhat different. This PaT-Net will be instantiated with the input parameter of the goal opposition type. The second controls the adduction (or abduction) of the fingers for grasps in which the fingers must be spread or pressed close together. Finally, there is a single PaT-Net which controls all of the thumb's movement for a particular grasp based on the opposition type.

2.4 Invoking the Grasp Behavior

While the Grasp behavior handles the execution of hand closure into particular grasps, other parameters required for realistic animation are missing. It is not responsible for transport of the hand to the object, and while palm orientation affects a grasp, selecting the orientation is the

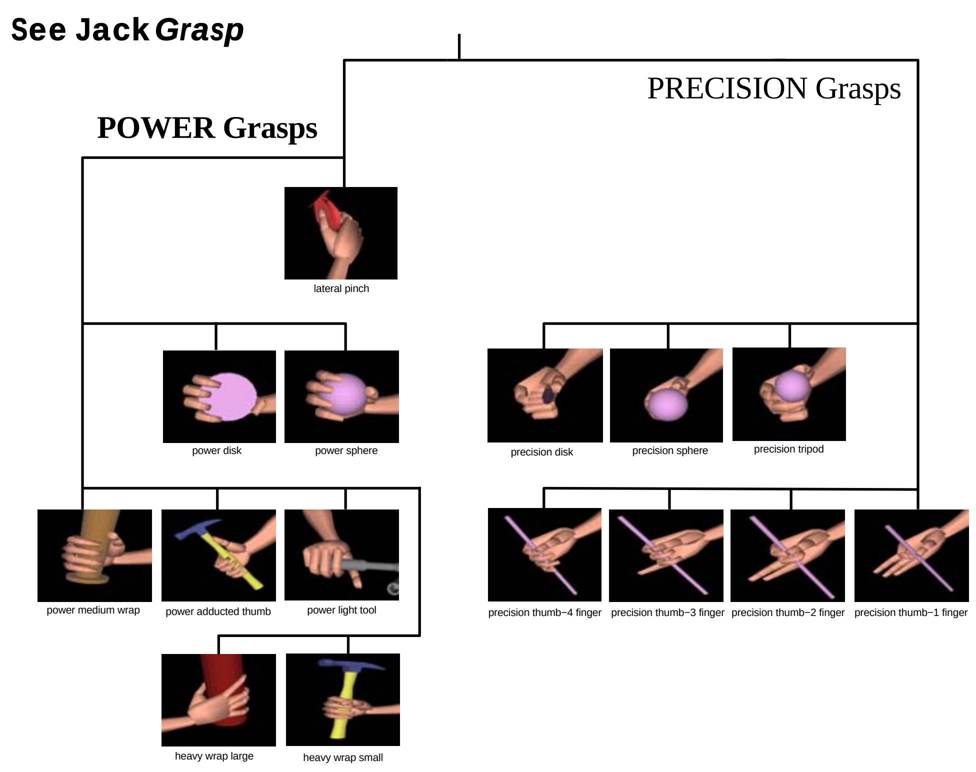

Figure 4: *Jack*'s Taxonomy of Manufacturing Grasps

responsibility of a higher-level process. Determining these parameters must be handled by an additional system, if not by the animator.

3 Object Specific Reasoning

General-purpose AI planners, in decomposing task plans into the steps which comprise the plan, produce a set of steps which might achieve the desired manipulation. These *task-actions* are unparameterized commands such as `(pickup jack hammer)` [1] or `(open jack door)`. While a human agent can interpret and perform such task-actions, they are underspecified for an animated agent.

[1]The first term indicates the action to take; the second term names the agent; the third term refers to the object of the action. Assume that agent and object names are uniquely identified with an entity in the animation scene; symbol grounding is beyond the scope of the present paper (cf, however, [GLM94]).

Robotic applications, meanwhile, use domain-specific manipulation procedures in which each motion is specified completely. These procedures are too specific to be generally useful; a system designed to remove four identical bolts securing a lid might require four separate bolt removal routines, due to the different locations of the bolts. Alternatively, systems may require human operators in the control loop: operators oversee operation and add situation-specific details to the motion directives.

The problem is further complicated by the fact that the high-level planner often ignores the environment in which the action is to occur: the high-level planner does not know enough about the physical description of an object and the relative positions of agent and object to accurately describe neither how to close the hand around the object nor where on the object to grasp. Likewise the simulator does not know enough about the purpose of the action to select the optimal parameters: grasping a hammer to use it, versus moving it, can result in different actions.

This suggests the need for an intermediate reasoning system which determines the unspecified parameters. The Object Specific Reasoner (OSR) [Lev95] breaks task-actions from the high-level planner into sets of motion directives, adding sufficient details such that the actions can be performed by existing behaviors controlled from the simulation system. The major contributions of the OSR are the recognition of the need for an intermediate reasoning model to map task-actions to motion directives, and the acknowledgement of the importance of the object and the purpose in interpreting a task-action.

Numerous high-level systems ignore the need for details to perform task-driven behaviors, and low-level systems have designed domain specific solutions which do not generalize. We have found few other systems developing general methods to build plans to manipulate objects; and none which formulate the problem solution as an intermediate reasoner. The issue is avoided by: 1) ignoring the fact that task-actions must be decomposed and parameterized; 2) selecting domains and tasks where manipulation is not crucial [VB90, LR90, McD93, Fir87, Str94, ZJ91]; or 3) concentrating on general manipulation strategies (e.g., the "peg in hole" problem) when the task is generically specified ([IJLZ88, LPMT84]).

3.1 OSR Motivation

The OSR maps task-actions to sets of motion directives. Its architecture is motivated by the following observations: that task-actions which specify the same task result in a wide variation of physical actions when applied to different objects in different environments. Consider the following task-actions:

(1) a. `(TApickup jack hammer (TAmove))`

b. `(TApickup jack glass (TAmove))`

c. `(TApickup jack papercup (TAmove))`

While the first two task-actions result in different behaviors, it is not clear that the last two will. Further, consider:

(2) a. `(TApickup jack crescent-wrench (TAmove))`

b. `(TApickup jack socket-wrench (TAmove))`

Finally, the *purpose* of a task-action affects the interpretation much as the object does: consider examples such as:

(3) a. `(TApickup jack hammer (TAmove))`

b. `(TApickup jack hammer (TAuse))`

In these examples the purpose of the task-action influences where an object is grasped and how it is moved.

The multiplicity of task-action/object pairs motivates the architecture for the OSR. The system must explain why the same task-action can appear with different objects. There are two possibilities: either a separate action plan definition is needed for each task-action/object pairing, or else task-action definitions are underspecified across some categorization of objects, and are complemented (completed) with additional object knowledge. The first alternative, of course, requires a

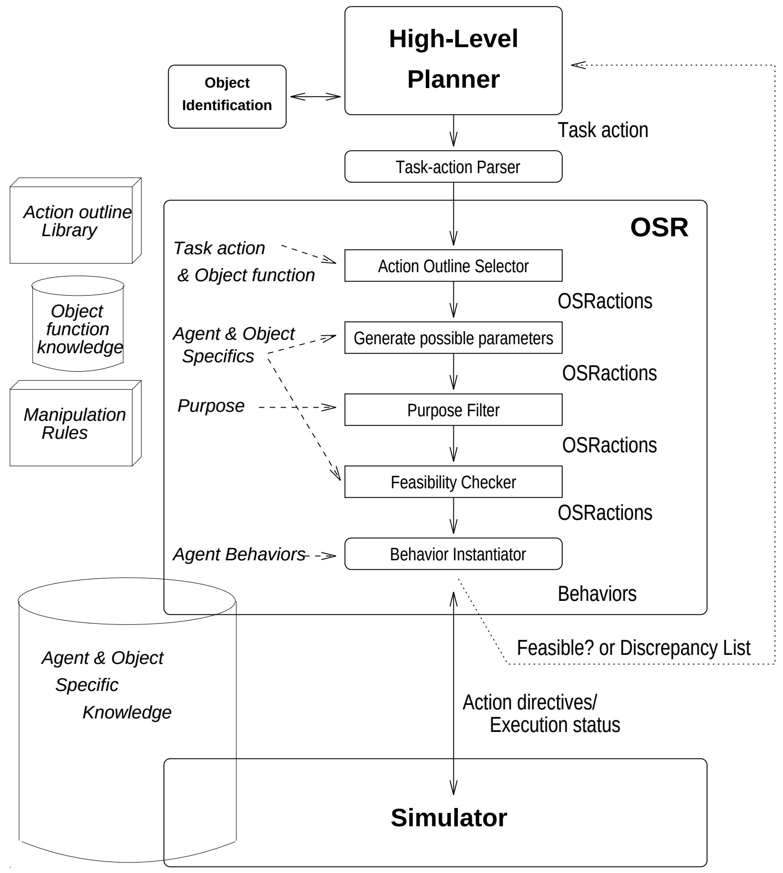

Figure 5: The Object Specific Reasoner

separate action definition for any task-action/object pairing that can occur; this is not a general-purpose solution. We adopt the second option here, as it addresses the variability of objects that can appear with a single task-action.

3.2 OSR Architecture

The Object Specific Reasoner expands existing task-action goals generated by a high-level planner and passes these plans to *Jack* for animation. The object manipulation task-actions are the primary input to the OSR. Given this input, the OSR performs two functions: 1) it can determine if a task-action is *feasible*, i.e., can be performed by the agent in the given context, and 2) it constructs the set of *motion directives* required to achieve the task-action.

Figure 5 illustrates the five phases of the OSR system: 1) selecting an *action outline*, 2) expanding all steps of the outline, 3) using details of the object of the current task-action to generate possible parameters for the motions in the outline, 4) selecting parameter values, and 5) verifying that the agent can perform this specific action on this specific object. If a set of motions can be so parameterized, the task-action is judged to be feasible.

In the first phase, the task-action and the type of the object are used to select an action outline from a library of outlines. This library is indexed by both the task-action and a taxonomy of object types. Objects are coarsely divided into geometric and functional categories: we differentiate, e.g., TOOLS, CLOSED CONTAINERS, OPEN CONTAINERS. For each task-action, there may be separate action outlines for each type of object that can appear in conjunction.

An action outline consists of a set of steps: each step is either a task-action or a *motion directive* – a direct call into the *Jack* behavioral system. As part of the first phase, action outlines are automatically expanded: sub-task-actions are replaced with their action outlines which are inserted into the master outline. This process continues until all task-actions are expressed by motion directives. Individual task-actions establish a partial ordering among their defining motions: whether two motions happen simultaneously, sequentially, or are dependent on each other. The final set of motions are partially ordered.

The second phase generates possible parameter values for all slots of the motion directives. Specific attributes of the agent and object, in conjunction with heuristic rules, are used to generate this set of values. This step moves the action outline from being a general plan to manipulate a generic object (of the specified category) to being a plan for a specific agent to manipulate a specific object.

The third phase sorts the possible parameter values and selects single parameters values to test. The *purpose* – the subsequent actions to be performed – are used to do this sorting. A set of heuristic rules are defined for each low-level motion; they are sensitive to object category and purpose. The sets of possible parameter values are sorted: for example, when `grasp`ing a TOOL to `use` it, the handle is the preferred grasp site; while when `grasp`ing a TOOL to `move` it, there is no preferred site (the system selects the closest accessible site). These rules are encoded as a set of object-oriented procedures.

The fourth phase involves checking dependencies between agent resources and object attributes. Each motion directive includes a predicate which specifies those pairs of resources and attributes to be checked. For example, the OSR might check whether the agent's hand is large enough to grip the handle of the hammer. If all the dependencies for all the motions in this outline are within tolerance, the OSR reports that the task-action is feasible.

If the agent and object attributes fail the tolerance test, then control is returned to the high-level planner along with a record, called a *discrepancy list*, of those resource/attribute pairs that are out of tolerance. The high-level planner can use this list to try to repair its plan.

If the set of motions passes the feasibility check, task-action refinement is complete, and agent-specific behaviors are substituted for the general motion language used by the OSR. These behaviors are sent to *Jack* for simulation. Any errors which occur during animation are relayed back to the high-level planner by the OSR for replanning or plan correction.

4 Examples

As an example, consider the task-action `(TAget jack hammer (TAuse))`[2]. (We take the meaning of `TAget` here to be "get control of" the object, i.e., reach to the object and grasp it.) Recall the syntax of a task-action: here, the agent jack is to get-control-of the object hammer for the purpose of moving it.

4.1 Example 1: `(TAget jack hammer (TAuse))`

The OSR begins by selecting an action outline for this task-action/object category: `TAGet`/TOOL.

- Phase 1: Select action outline.

[2] *TA* is an abbreviation for task-action; *OSR* indicates a primitive motion in the reasoning system.

```
SEQ
  (TAReach agt Tool))
  (OSRGrasp agt OpenCont)
```

- Phase 2: Expand outline, add agent, object and purpose specifics.

```
SEQ
  WHILE ((OSRReach Jack hammer (OSRGrasp TAUse))
        (OSRLook Jack hammer (OSRGrasp)))
  (OSRGrasp Jack hammer (TAMove))
```

- Phase 3: Generate possible parameter values.

```
SEQ
  WHILE ((OSRReach Jack {left right both} hammer
            {.head .base .handle} (OSRGrasp TAUse))
        (OSRLook Jack hammer (OSRGrasp))))
  (OSRGrasp Jack {left right both} hammer
            {.head .base .handle} {power precision clamp} (TAUse))
```

- Phase 4: Sort possible values; select one.

```
SEQ
  WHILE ((OSRReach Jack right hammer.handle)
        (OSRLook Jack hammer))
  (OSRGrasp Jack right hammer.handle power-adducted-thumb)
```

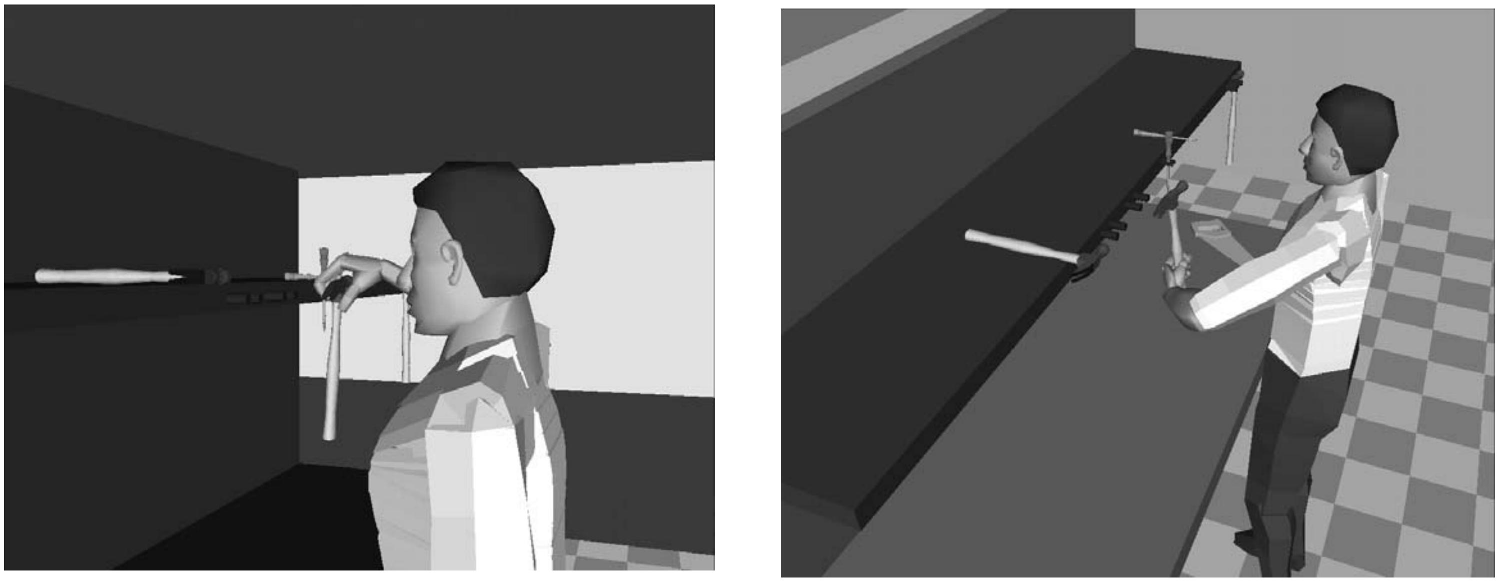

Figure 6: Grasping the hammer to move or to hammer.

- Phase 5: Check agent resources against object attributes for each OSRmotion. If they are within tolerance, the task-action is *feasible*.

When the OSR finishes successfully constructing the manipulation plan, the simulation system is invoked. In this example, a PaT-Net would be constructed which builds the dependency between the OSRReach and the OSRLook, and sequences that with the OSRGrasp.

When the OSRGrasp is invoked, an instance of the Grasp PaT-Net is created and begins to execute. The Grasp PaT-Net creates five sub-nets, one for each finger. These nets run simultaneously, moving the fingers individually around the object.

4.2 Example 2: `(get jack hammer (move))`

Processing in the first two phases is the same as the previous example. The major differences occur in Phase 4, when the purpose is used to sort possible parameters. Note in Phase 3 below the differences in the purpose sets between this and the `(get Jack hammer (use))` example.

- Phase 3: Generate possible parameter values.

```
SEQ
  WHILE ((OSRReach Jack {left right both} hammer
             {.head .base .handle} (OSRGrasp TAMove)))
        (OSRLook Jack hammer (OSRGrasp))
  (OSRGrasp Jack {left right both} hammer
             {.head .base .handle} {power precision clamp} (TAMove))
```

- Phase 4: Sort possible values; select one.

```
SEQ
  WHILE ((OSRReach Jack left hammer.head))
        (OSRLook Jack hammer)
  (OSRGrasp Jack left hammer.head clamp)
```

These differences are presented in Figure 6.

5 Conclusion

We have presented a general-purpose system to construct animations of a simulated human character manipulating objects. Our two-level architecture consists of an intermediate level reasoner which parameterizes motor behaviors programs, and a low-level system which actually executes the fine motor control.

Figure 7 illustrates the hierarchical structure of PaT-Nets in the system. An action node in one component expands to a full PaT-Net at the next level of detail. In the first expansion, the Object Specific Reasoner generates parameters for underspecified task-level instructions, examines task feasibility, and tests possible parameterizations of motion directives. A second expansion instantiates the Grasp behavior, which manages simultaneous motions of the joints in the hand,

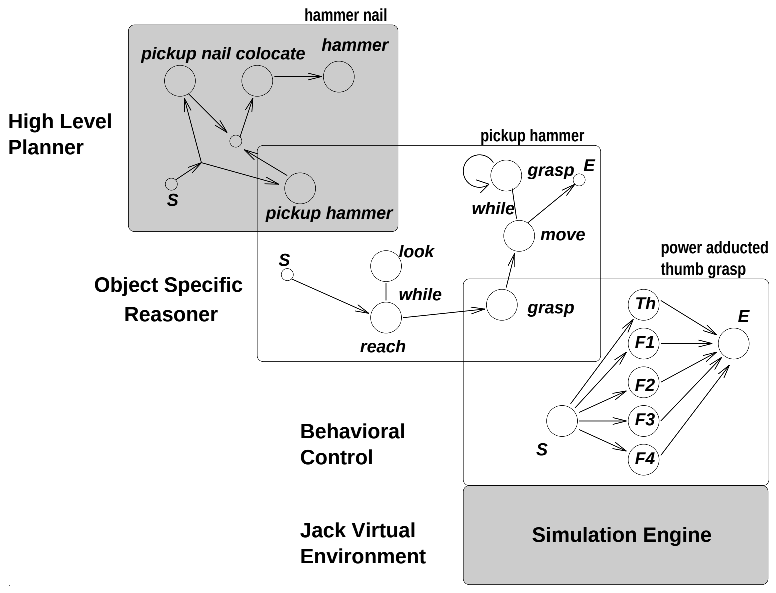

Figure 7: Hierarchical PaT-Net decomposition of a pickup task.

wrist and arm. When composed hierarchically, the OSR and the Grasp behavior allow the animator task-level control over the simulated humans in the virtual environment.

The OSR's ability to interpret task-actions with respect to the object and context allows an animator to use a single task-action to describe the action, without concern for situation specific details. Similarly, the behavioral simulator allows the OSR to discuss motions, e.g., `grasp` without concern for fine motor control. The Grasp behavior provides the OSR with a succinct interface to Grasp behaviors; the OSR provides an animator a succinct interface to Grasping task-actions. Operating together, these two separate systems combine to provide a powerful tool for a high-level planner or animator to use when generating realistic animations. This system is implemented as part of the *Jack* project at the University of Pennsylvania.

The ease with which we linked the OSR and the Grasp behavior leads us to believe that our future work to include other agent behaviors looks promising. The two levels of expansion between the planner and the virtual environment allows each component to attack smaller, less cumbersome problems, while remaining independent: the Grasp behavior can be improved without affecting the functionality of the OSR, and vice versa. This flexible architecture will support the incorporation of additional object manipulation behaviors, allowing us to extend this task-level interface to a virtual environment.

6 Acknowledgements

This research is partially supported by ARO DAAL03-89-C-0031 including U.S. Army Research Laboratory, and ARO DURIP DAAH04-95-1-0023; ARPA AASERT DAAH04-94-G-0362; DMSO DAAH04-94-G-0402; ARPA DAMD17-94-J-4486; U.S. Air Force DEPTH through Hughes Missile Systems F33615-91-C-0001; DMSO through the University of Iowa; NSF IRI 91-17110 and CISE CDA88-22719.

We thank Diane Chi and Matthew Stone for their help with this paper.

References

[AIL85] M. A. Arbib, T. Iberall, and D. M. Lyons. *Coordinated control programs for movements of the hand*, pages 111–129. Springer-Verlag, Berlin, 1985.

[BB93] Welton Becket and Norman I. Badler. Integrated behavioral agent architecture. Conference on Computer Generated Forces and Behavior Representation, 1993.

[Bec94] Welton Becket. The Jack LISP API. Technical Report MC-CIS-94-01, University of Pennsylvania, 1994.

[BLTK93] George A. Bekey, Huan Liu, Rajko Tomovic, and Walter J. Karplus. Knowledge-based control of grasping in robot hands using heuristics from human motor skills. *IEEE Transactions on Robotics and Automation*, 9(6):709–722, 1993.

[BPW93] Norman I. Badler, Cary B. Phillips, and Bonnie L. Webber. *Simulating Humans: Computer Graphics Animation and Control.* Oxford University Press, 1993.

[CH90] Mark R. Cutkosky and Robert D. Howe. Human grasp choice and robotic grasp analysis. In S. T. Venkataraman and T. Iberall, editors, *Dextrous Robot Hands*, pages 5–31. Springer-Verlag, New York, 1990.

[CPB$^+$94] Justine Cassell, Catherine Pelachaud, Norm Badler, Mark Steedman, Brett Achorn, Tripp Becket, Brett Douville, Scott Prevost, and Matthew Stone. Animated conversation: Rule-based generation of facial expression, gesture and spoken intonation for multiple conversational agents. In *SIGGRAPH*, 1994.

[Fir87] R. James Firby. An investigation into reactive planning in complex domains. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 202–206, 1987.

[GLM94] Christopher Geib, Libby Levison, and Michael B. Moore. Sodajack: an architecture for agents that search for and manipulate objects. Technical Report MS-CIS-94-13, University of Pennsylvania, 1994.

[IBA86] T. Iberall, G. Bingham, and M. A. Arbib. Opposition space as a structuring concept for the analysis of skilled hand movements. In H. Heuer and C. Fromm, editors, *Generation and modulation of action patterns*, pages 158–173. Berlin: Springer-Verlag, 1986.

[Ibe86] Thea Iberall. The representation of objects for grasping. In *Proceedings of the Eighth Cognitive Society Conference*, pages 547–561. Cognitive Society, 1986.

[Ibe87a] Thea Iberall. A ballpark approach to modelling human prehension. In *1987 IEEE International Conference on Neural Networks*, volume IV, pages 535–543. IEEE, 1987.

[Ibe87b] Thea Iberall. Grasp planning for human prehension. In *IJCAI*, pages 1153–56, 1987.

[IJLZ88] Thea Iberall, Joe Jackson, Liz Labbe, and Ralph Zampano. Knowledge-based prehension: Capturing human dexterity. In *IEEE Intl. Conf. on Robotics and Automation*, pages 82–87, 1988.

[KKKL94] Yoshihito Koga, Koichi Kondo, James Kuffner, and Jean-Claude Latombe. Planning motions with intentions. In *Computer Graphics (SIGGRAPH '94 Proceedings)*, pages 395–408, 1994.

[Lev95] Libby Levison. Connecting planning and acting: Towards an architecture for object specific reasoning. disseration proposal, 1995.

[LPMT84] Tomás Lozano-Pérez, Matthew. T. Mason, and Russell H. Taylor. Automatic synthesis of fine-motion strategies for robots. In Michael Brady and Richard Paul, editors, *Robotics Research*, pages 65–95. MIT Press, 1984.

[LR90] John Laird and Paul Rosenbloom. Integrating execution, planning and learning in SOAR for external environments. In *Proceedings of the 8^{th} National Conference on Artificial Intelligence*, pages 1022–1029, 1990.

[McD93] Drew McDermott. Transformational planning of reactive behavior. Technical Report RR-941, Yale University, 1993.

[MGR95a] Michael B. Moore, Christopher Geib, and Barry D. Reich. Planning and terrain reasoning. In *AAAI Spring Symposium on Integrated Planning Applications*, 1995. (also University of Pennsylvania CIS department Technical Report MS-CIS-94-63/LINC LAB 280).

[MGR95b] Michael B. Moore, Christopher Geib, and Barry D. Reich. Planning for reactive behaviors in hide and seek. In *Proc. 5th Conference on Computer Generated Forces and Behavioral Representation*, Orlando, FL, May 1995. Institute for Simulation and Training.

[MI94] Christine L. MacKenzie and Thea Iberall. *The Grasping Hand*, volume 104 of *Advances in Psychology.* North-Holland, 1994.

[MML90] R. G. Marteniuk, C. L. MacKenzie, and J. L. Leavitt. The inadequacies of a straight physical account of motor control. In H. T. A. Whiting, O. G. Meijer, and P. C. W.

van Wieringen, editors, *The Natural-Physical Approach to Movement Control*, pages 95–115. Ablex, Norwood, NJ, 1990.

[MTN88] Thalmann D. Magnenat-Thalmann N., Laperriere R. Joint-dependent local deformations for hand animation and object grasping. In *Proceedings of Graphics Interface '88*, pages 26–33, June 1988.

[MTT91] Nadia Magnenat-Thalmann and Daniel Thalmann. *Making Them Move: Mechanics, Control, and Animation of Articulated Figures*, chapter Human Body Deformations Using Joint-dependent Local Operators and Finite Element Theory, pages 243–262. Morgan Kaufmann, 1991.

[Nap93] John Napier. *Hands.* Princeton University Press, 1993. Revised by John Tuttle.

[Pia53] Jean Piaget. *The Origins of Intelligence in Children.* London: Routledge & Kegan Paul, 1953.

[RG91] Hans Rijpkema and Michael Girard. Computer animation of knowledge-based human grasping. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 339–348, July 1991.

[Sta90] S. A. Stansfield. Knowledge-based robotic grasping. In *Proceedings of the 1990 IEEE International Conference on Robotics and Automation*, pages 1270–1275. IEEE, 1990.

[Str94] Steve Strassman. Semi-autonomous animated actors. In *Proceedings of the* 12^{th} *National Conference on Artificial Intelligence*, pages 128–134, 1994.

[TBK87] Rajko Tomovic, George A. Bekey, and Walter J. Karplus. A strategy for grasp synthesis with multi-fingered robot hands. In *IEEE Intl. Conf. on Robotics and Automation*, pages 83–89, 1987.

[VB90] Steven Vere and Timothy Bickmore. A basic agent. *Computational Intelligence*, 6:41–60, 1990.

[ZJ91] David Zeltzer and Michael B. Johnson. Motor Planning: an Architecture for Specifying and Controlling the Behavior of Virtual Actors. *The Journal of Visualization and Computer Animation*, 2:74–80, 1991.