Embodied Object Schemas for Grounding Language Use

by

Kai-yuh Hsiao

M.Eng., Massachusetts Institute of Technology (2001)
S.B., Massachusetts Institute of Technology (1999)

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Media Arts and Sciences

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2007

Author ____
Program in Media Arts and Sciences
April 30, 2007

Certified by ____
Deb K. Roy
Associate Professor of Media Arts and Sciences
Program in Media Arts and Sciences
Thesis Supervisor

Accepted by ____
Andrew B. Lippman
Chair, Departmental Committee on Graduate Students
Program in Media Arts and Sciences

Embodied Object Schemas for Grounding Language Use

by

Kai-yuh Hsiao

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
on April 30, 2007, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Media Arts and Sciences

Abstract

This thesis presents the *Object Schema Model* (OSM) for grounded language interaction. Dynamic representations of objects are used as the central point of coordination between actions, sensations, planning, and language use. Objects are modeled as *object schemas*—sets of multimodal, object-directed *behavior processes*—each of which can make predictions, take actions, and collate sensations, in the modalities of touch, vision, and motor control. This process-centered view allows the system to respond continuously to real-world activity, while still viewing objects as stabilized representations for planning and speech interaction. The model can be described from four perspectives, each organizing and manipulating behavior processes in a different way. The first perspective views behavior processes like thread objects, running concurrently to carry out their respective functions. The second perspective organizes the behavior processes into object schemas. The third perspective organizes the behavior processes into plan hierarchies to coordinate actions. The fourth perspective creates new behavior processes in response to language input. Results from interactions with objects are used to update the object schemas, which then influence subsequent plans and actions. A continuous planning algorithm examines the current object schemas to choose between candidate processes according to a set of primary motivations, such as responding to collisions, exploring objects, and interacting with the human. An instance of the model has been implemented using a physical robotic manipulator. The implemented system is able to interpret basic speech acts that relate to perception of, and actions upon, objects in the robot's physical environment.

Thesis Supervisor: Deb K. Roy
Title: Associate Professor of Media Arts and Sciences, Program in Media Arts and Sciences

Embodied Object Schemas for Grounding Language Use

by

Kai-yuh Hsiao

The following people served as readers for this thesis:

Thesis Reader____

Dr. Bruce Blumberg
Senior Scientist
Blue Fang Games

Thesis Reader____

Rodney A. Brooks
Professor of Robotics
Massachusetts Institute of Technology

Acknowledgments

Many thanks to all the people who have directly and indirectly contributed to both the project and to my progress through graduate school. Most importantly, I believe Deb Roy is the most unique advisor I could ever have come across and I consider myself greatly fortunate to have met him at the time that I did. From the beginning, his research agenda was the only one I came across that seemed interested in "doing it right." Since then, he has worked tirelessly to bring in the people and resources necessary to build what I consider to be a visionary set of research platforms, without losing sight of the big picture. Deb's continued support of my efforts, despite the occasional hurdle and setback, has made all the difference on this path, and I owe him an enormous debt of gratitude.

Thanks also to my committee members, Bruce Blumberg and Rod Brooks, for their support and advice, and especially for their patience with my meanderings and perpetually changing timeline.

And, of course, any large coordinated project is practically impossible to achieve nowadays without whole teams of people working alongside each other to get things up and running. I owe a whole lot of people for contributing months and months of their time for helping to build the system. It seems almost unfair to condense their efforts into a few lines of text, since it belies the sheer dedication and painstaking work required for every single piece of the system:

- Peter Gorniak, for building the parser, doing the initial integration of the speech recognizer, and being a generally cool officemate.
- Jonathan Huggins, for helping to build and test the robot arm controller and the initial graphical interfaces.
- Heather Knight, for designing Trisk's head.
- Rony Kubat, for bringing up the neck and eye controllers, interfacing them to the rest of the system, and building all the cool graphical interfaces that he's so good at making.
- Peter Lai, for his work on robot arm control and building the mean-shift tracker.
- Nikolaos Mavridis for building the mental model in the Ripley system, part of our earlier efforts at robotic language use.
- John McBean, for his exemplary design and construction work on the physical robot hardware.

- Kailas Narendran, for his electrical work on the robot components and for his subsequent support role in robot control debugging.
- Philipp Robbel, for his work adapting, building, extending, and training the visual segmenter and shape recognizer.
- Stefanie Tellex, for her work in adapting the speech and parsing modules to the new system, and for slogging painfully through my code to help build the interfaces between words and structures within the model.
- Soroush Vosoughi, for staying up all night, time and time again, to get innumerable aspects of the visual systems, the robot controllers, the network interfaces, and the final integration working.
- David Wang, for building the 3-D arm visualization model and integrating the robot hand controllers.
- Jonathan Wang, for his work on getting initial control and state information from the robot hand.

Additional thanks to everyone else from the Cognitive Machines group, for their advice, support, and camaraderie over the years, and especially our administrators, Kristie Thompson, Tamara Hearn, Nadia Ustariz, and Mutsumi Hata, who have kept things running smoothly against all odds. And finally to all my friends and family for keeping me alive and sane over the years.

Contents

List of Figures

Chapter 1

Grounded Language, Reactive Behavior, and Object Schemas

Computers are frequently used to store and process words. However, within a typical computer, the presence of a word such as “cup” does not relate to any sensorimotor notion of real-world cups. For any normal computer program, the connection between a word and its real-world referent is supplied only by the human reading the words. In order for words in a computer to connect to structures or processes directly related to the real world, the software will need at some point to have access to sensor data and possibly motors.

Robots can provide sensors and motors for a computational system. Putting the system in a real-world body and designing it to deal with the body’s physical properties makes it *embodied*. However, simply using a robot is not sufficient for connecting words within the system to perceptions and actions in the real world. The processes and representations within the robot system need to be organized in a way that reflects the structure of the external world, while providing stability for the connection to language amid noise and changes in the environment.

The effort to connect language to the world via sensors and sometimes motors is called *language grounding*. Examples of grounded language systems are described in [4, 6, 7, 48, 56, 63, 69, 68, 74, 75, 82].

For a language-using robot to be useful in a human environment, it must do more than interpret words. While carrying out assigned tasks, a robot has to adjust its actions in light of sudden changes in the environment or the task. This requires the sorts of reactivity demonstrated in *behavior-based robotics*, which includes systems such as in [14, 15, 17, 18, 30, 31]. Behavior-based design attempts to maximize the connection between the real world and the decision-making of the robot system. This is achieved by limiting the complexity of internal representations, using rapid sensorimotor loops with minimal state, and reacting directly to sensor inputs whenever possible.

In order to use language, it is not possible to completely eliminate internal representations. A phrase such as "the cup" refers to a construct that has its roots in some aspect of the external environment (i.e., a cup near the robot), but it must continue to refer to the same object even when the robot can no longer see the cup, whether due to occlusion or sensor noise. This requires internal representation. Thus, the best reactivity possible for a language-using robot is achieved by using an internal representation of objects and actions that stays in contact with current sensor and motor states as much as possible, while reacting immediately to sensors when necessary, such as for collision avoidance.

1.1 Object Schemas for Representation and Integration

The key idea of this thesis is the *object schema* as an internal representation of concrete, physical objects that enables integration between language, action planning, and the continuous *modalities* of motor action, vision, and touch. Each object schema consists of a set of *behavior processes* (or just *processes*) and an *interaction history.* Behavior processes act like thread objects in a multithreaded program, running concurrently with the rest of the system and retaining a small amount of internal state. Interaction histories serve as data storage, holding data and outcomes related to the behavior processes of an object schema. Each behavior process writes data to the interaction history of its object schema in a programmed way, and other behavior processes can then read the written data. Some behavior processes read sensor data, others issue motor commands, and others just read and write interaction history data.

The main purpose of this document is to explain the *Object Schema Model* (*OSM* for short), which uses behavior processes and object schemas to integrate actions, sensing, planning, and language. I will use the "OSM" abbreviation both for brevity and to distinguish this model from other models that also make use of object schemas. The secondary purpose of this document is to describe the use of the OSM in a complete integrated robotic system that interacts with the world via motor action, vision, touch, and speech.

The main contribution of this thesis is the use of the Object Schema Model to integrate aspects of language grounding systems with aspects of behavior-based systems. To provide a foundation for language grounding, object schemas provide stable identities that reflect perceived objects, and are organized in a way that makes language about object-directed actions convenient. To enable behavior-based reactivity, the behavior processes run concurrently to handle sensing and motor control.

Conceptually, the OSM represents an object as the sum of the processes that act on it, instead of passively viewing an object as a set of sensory attributes. This connects the representation of an object more directly to the embodiment of the system, emphasizing the interactivity that characterizes behavior-based design while still providing stable constructs for language and planning.

Within this primary contribution of integration, several other properties of the OSM constitute secondary contributions. Because object schemas are composed of behavior processes, data about an object is shared rapidly between processes, allowing for integration between continuous modalities such as touch and vision. Using multiple modalities helps attenuate the effects of sensor noise, by offering multiple methods of perceiving and verifying the identity of an object. The sharing of data, and the additional use of behavior processes as components of the action planning system, also enables opportunistic planning, in which new information from the environment, such as successes and failures of actions, perceptions about objects, and verbal inputs, can immediately change the planning of actions subsequently taken by the system. Generally, the central role of object schemas and behavior processes enables the system's reactivity, actions, sensor data, planning, and language to continuously inform and shape one another.

In order to explain the motivation and operation of some key aspects of the object schema representation, I will first briefly describe the robot platform on which the OSM

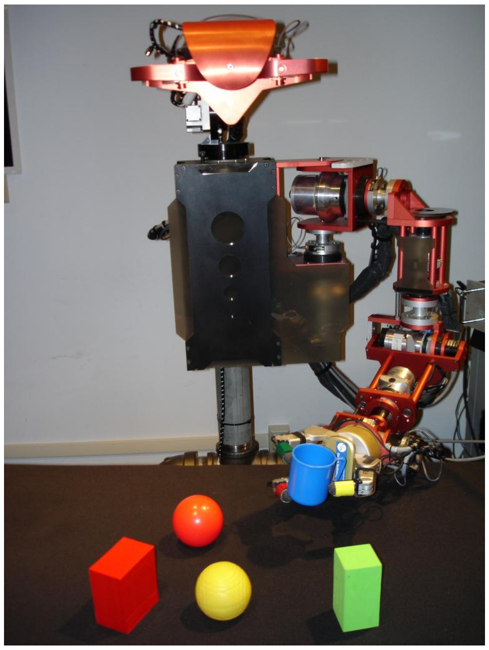

Figure 1-1: Trisk is a humanoid upper-body robot with 4-DOF (degree-of-freedom) neck, 6-DOF arm, 4-DOF hand, and two cameras in the head. It is mounted facing a table, on which it can perceive and manipulate objects.

is implemented. Then, I will narrate some sample interactions of the implementation and some example properties of the OSM demonstrated by the interactions.

1.1.1 Trisk, a Robotic Manipulator

Trisk is a humanoid upper-body robot with one arm, one hand, a neck, and a head. It is mounted in a fixed position facing a table, on which objects can be placed for perception and manipulation. Force-torque sensors in each of the three fingers of the hand enable sensing of touch contact points. Two cameras in the head provide visual input.

Visual input from one of the cameras (stereo vision was implemented but then left out for simplicity) is segmented into regions of uniform color. A mean-shift tracking process can additionally be assigned to track any single region in subsequent frames based on color, location, and edge profile. Regions are also used to produce shape and color categories.

The robotic hand can be used to grasp and move objects. The success and failure of a grasp can be detected based on feedback from the touch sensors. Excessive forces from the touch sensors in the fingers and the load cells at each arm joint are used to detect collisions.

1.1.2 Interaction 1: Exploring Objects

Next, I will describe a few example interactions between the robot and its environment, and some of the properties of the model demonstrated by the interactions. The mechanisms behind the interactions will be explained in greater detail in later chapters; the point of these examples is to give an overall sense of the structure of the system.

The first interaction:

> *Two objects, a lightweight green block and a heavy lantern battery, are placed on the table. Trisk reaches for each object, lifts it, and puts it back down. After doing so, the system notes internally that the green block is lightweight, green, and rectangular, and the battery is heavy, gray, and rectangular.*

The visual regions resulting from the color-based segmentation of the camera input lead to new object schemas, one for each region. Recall that object schemas consist of behavior processes and interaction history data. Each object schema in this example consists initially of a visual tracking process (or just "visual tracker") that loops continuously, looking at each subsequent color-segmented frame to find regions that are signs of the same object. An object schema such as the one being described throughout this section is illustrated in Figure 1-2.

Visual attributes observed by the visual tracker, such as color, size, and centroid location (in two-dimensional *visual coordinates* as seen by the cameras, as opposed to three-dimensional *arm coordinates*, which are target locations for the robot hand) are written to the interaction history of its object schema. The data written to the interaction history by the trackers and other processes are called *attributes.* The *attribute field* is the category of the data, such as color, size, shape, and coordinates. The *attribute value* is the value of the attribute field for that specific object schema.

In addition to the visual tracker, other processes are added to the object schema to continuously translate the data from the visual tracker into categorized colors, shapes, and arm coordinates. Categorical information serves as discrete identifiers that can connect to words in the speech input. The categories are *semi-stable* in that they provide discrete tokens that tend to persist over time, but can be changed rapidly if the continuous attributes

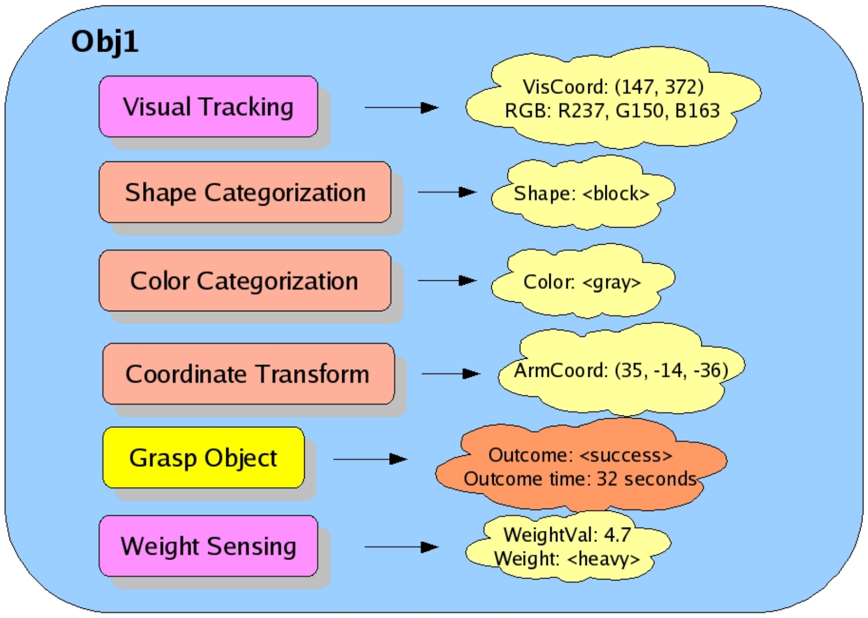

Figure 1-2: An illustration of the object schema described in Interaction 1. Small boxes are the behavior processes belonging to the object schema labeled “Obj1.” Clouds represent the interaction history data that has been written by the behavior processes. Yellow clouds represent attributes of the object schema gathered by the processes, and orange clouds represent outcome data of processes attempting to target the object. “Color,” “Shape,” “Weight,” etc. are called *attribute fields*, and their associated values are called *attribute values*. The different box colors represent sensory processes (for “visual tracking” and “weight sensing”), translation processes (for “shape categorization,” “color categorization,” and “coordinate transform”), and action processes (for “grasp object”).

of the object are sufficiently altered due to the perceptual inputs.

In the absence of collisions and requests from the human, Trisk explores objects by reaching for them and attempting to lift them. Each grasp attempt is governed by a series of behavior processes that read the object's expected location and send appropriate motor commands to reach towards, grasp, and lift the object. When a grasp is successful, a touch-based tracking behavior process (or just "touch tracker") continuously writes a location attribute to the object schema's interaction history based on the current location of the robot hand. This makes use of the assumption that while an object is being grasped by the robot, the object's location is the same as the hand's location.

In addition to attributes, the other kind of data written to interaction histories are *outcomes*, which are written whenever a behavior process completes or fails. The processes for reaching, grasping, and lifting all have criteria that evaluate success or failure of their actions, and when these processes succeed or fail, they write whether it was a success or a failure (the *outcome*) and elapsed running time of the process (the *outcome time*) to the interaction history.

As the robot lifts each object, the force sensors detect not only a grasping force to confirm the continued success of the grasp, but they also detect the downwards force on the fingers, which constitutes a measure of weight. An additional sensory process writes this weight attribute to the interaction history.

The main purpose of this example is to explain that each object schema consists of continuously-running behavior processes and their interaction history data, and to introduce the idea of attributes and outcomes as the two types of interaction history data.

1.1.3 Interaction 2: Using Outcome Data to Plan a "Group" Action

The human issues the command, "Group the green block and the gray block." Trisk reaches for the battery (the "gray block") and attempts to move it towards the green block. As it lifts the battery, the heavy battery slips out of the robot's hand. Trisk notices this and immediately opts to lift the green block and move it near the battery instead.

In Interaction 1, the robot reached for each object in order to explore its properties.

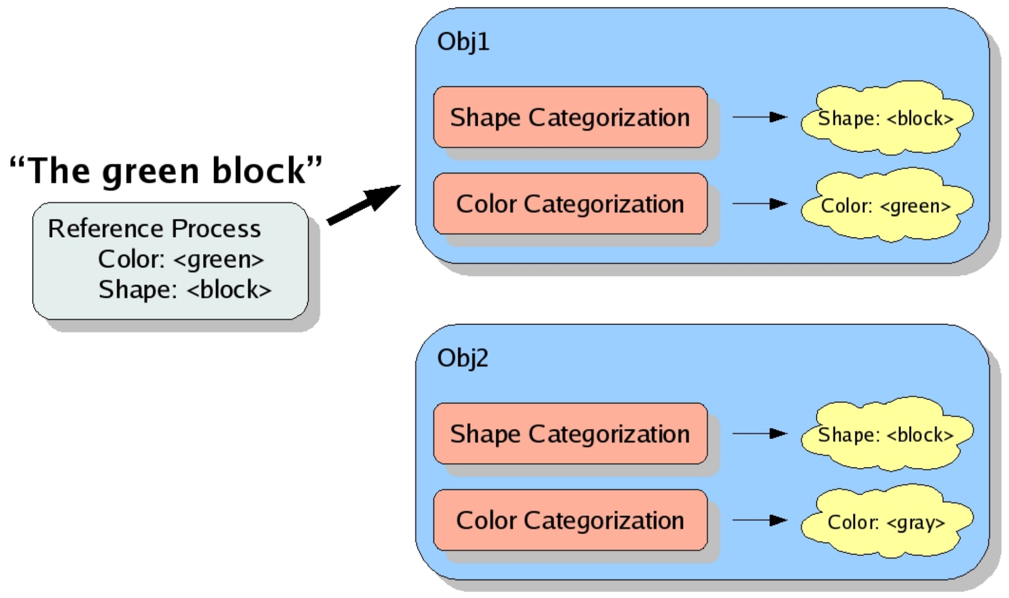

Figure 1-3: On the left is a rectangle representing the reference process assigned to the noun phrase "The green block." On the right is a depiction of two object schemas (each showing only two of their behavior processes) with specific attributes in their interaction histories. The reference process matches itself to the appropriate object schema, and specified actions can then be directed towards that object schema.

The behavior processes that govern these actions are instantiated (created) by the "curiosity motivation," one of three *primary motivations* that instantiate and support behavior processes relevant to their respective purposes. The second primary motivation is the motivation to respond to verbal requests from the human. Behavior processes generated based on verbal inputs are instantiated and supported by this "social motivation."

Verbal commands are received via a wireless headset microphone. Speech recognition is performed on the audio input and the word stream is then parsed into a grammatical parse tree. The tokens in the parse tree include *verb tokens*, *noun phrase tokens*, and the *description tokens* that are constituents of the noun phrase token (e.g., "green" and "block" are description constituents of the complete noun phrase "the green block"). The noun phrases refer to objects, and thus the noun phrase token must be connected to an object schema representing the appropriate object. This is accomplished by assigning a *reference process* to the noun phrase token, which matches object schemas based on the current set of attributes for each object schema. In this example, the reference process for “the green block” matches the object schema Obj1, based on its categorical attributes (see Figure 1-3). Once the reference processes are matched, any actions specified as a result of the verb token can be targeted towards the matched object schemas.

The verb token associated with the verb “group” leads to the creation of a *plan fragment process* (or just *plan fragment*), which coordinates a sequence of *action processes* (or just *actions*, in the proper context) and *condition processes* (or just *conditions*, in the proper context) for use in planning. The plan fragment process in this example specifies a condition that the two objects named in the verbal input must be moved close to each other. This condition in turn can be satisfied by moving either object towards the other one. Each of these two possibilities is represented by another plan fragment process, each coordinating the actions to move one of the objects towards the other. Notice that plan fragments, actions, and conditions are behavior processes, meaning that they too are components of object schemas, running concurrently with each other and writing data to the interaction history of their targeted object schema.

These particular types of behavior processes are used for planning actions. The initial plan fragment for “group” is supported by the social motivation. This support leads to the creation of a plan and eventually to the execution of relevant actions. The plan involves connecting the plan fragment and condition processes in a structure called a *plan hierarchy*, in which plan fragments connect to their specified actions and conditions, and conditions in turn connect to plan fragments that will lead to their satisfaction. This hierarchy, with the primary motivations, is depicted in Figure 1-4.

In building the plan hierarchy, the model must decide between the two plan fragments that satisfy the condition. This decision is made based on the likelihood of success of the two possible plan fragments. When the battery slips out of the fingers, the behavior process responsible for lifting the battery notices the cessation of the grasping force on the fingers. This leads to a failure outcome being written to the interaction history, which in turn leads

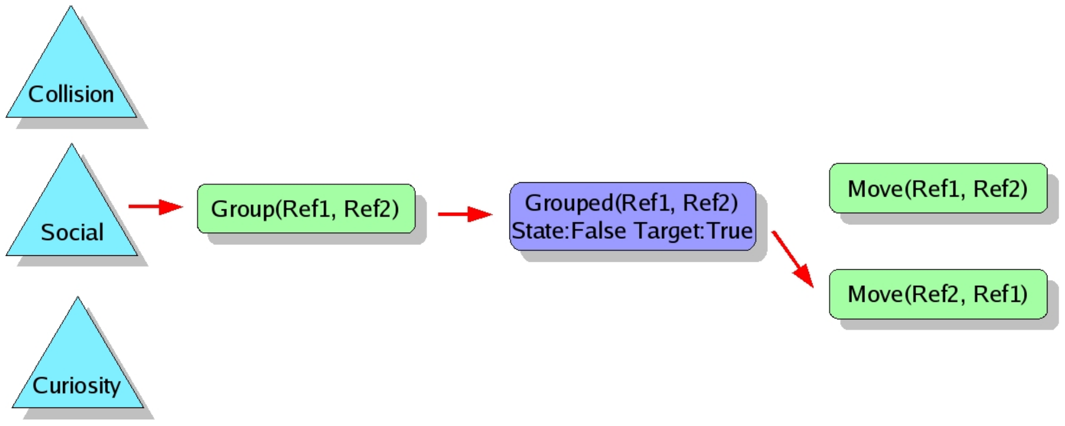

Figure 1-4: Depiction of the planning system. Plan fragments are in green, conditions are in blue, and primary motivations are represented by triangles. On the left are the three primary motivations. The social motivation is supporting a plan fragment to perform the task discussed in Interaction 2. The plan fragment to group two objects, Group, specifies a condition, Grouped, which can be made true by either of two plan fragments of type Move. The red arrows denote support (*priority scores*, discussed in Chapter 3) coming from the social motivation that leads to the construction of the hierarchy and execution of the actions.

to a re-evaluation of the success likelihoods. Because of the failure of the lifting process in the battery's object schema, the lifting process in the green block's object schema now has a higher likelihood, and is selected to satisfy the condition instead.

Object schemas are not the only structures with interaction histories. Additional histories are also stored for each plan fragment process. When the attempt to lift the heavy battery fails, an additional piece of outcome data is written notating a failure for the `lift` plan fragment with respect to a heavy object. All the object schemas, as well as interaction histories for object schemas and plan fragments, are stored in the *belief context*, which constitutes the system's set of beliefs about the current environment and expected outcomes of actions.

This example is used to discuss language use and planning. It also shows that outcome information written to the interaction histories is used to evaluate decisions in the planning process.

1.1.4 Interaction 3: Using Attribute Data to Plan a "Group" Action

The human adds a red apple to the tabletop and issues the command, "Group the green block and the red apple." Trisk reaches for the red apple, but before grasping it, the human then says, "The red apple is heavy." Trisk immediately backs away from the red apple, and opts to lift and move the green block instead.

The decision-making process to choose between the two plan fragments is the same here as in Interaction 2, but this time the human provides additional verbal information about the red apple, which the robot has not had a chance to explore. When the red apple is described as being "heavy," the attribute is written to the interaction history and used in the same way as if it had been written due to an actual lift action. In this case, the history for the `lift` plan fragment, which includes the data from Interaction 2, is used to decide that object schemas with the `heavy` attribute have a lower success likelihood for the `lift` plan fragment than the default.

Thus, by verbally supplying an object attribute, the human provides information that the robot can opportunistically use to alter its plan to improve the likelihood of achieving its goal. The prediction of the outcomes and effects of actions taken towards an object amounts to representing what is called the *affordances* of an object, a term coined by J. J. Gibson [33] to describe what actions, abilities, and states an object enables for an agent.

Attributes Connect Language, Planning, and Sensorimotor Interaction

The main purpose of this example is to show that object attributes, which normally derive from direct visual and tactile interaction with objects, can also derive from language, and furthermore can connect to the planning process.

This is especially interesting because other models of language grounding often connect the perceived attributes of an object to words, which is a challenging task in itself. However, simply connecting attributes to language ignores the question of why those attributes would have been useful to collect and talk about in the first place. Sensorimotor object attributes are useful for the goals of a system because they relate to expected outcomes of actions taken on objects. A large object is hard to grasp. A heavy object is hard to lift. In biological domains, a red object may be poisonous. A single attribute of an object, such as size, weight, and color, can relate to outcome expectations for numerous actions that could be taken towards an object.

Thus, one key point of the OSM is to establish object attributes as a means of informing outcome expectations for the purpose of planning. By connecting the attributes to language as well, it makes verbally-derived attributes (such as from "The red apple is heavy") useful towards planning for the current goals of the system, rather than just as a means of referring to objects.

Process-Centered Schemas Enable Sharing and Affordance Representation

Another key point of the OSM shown in this example is to provide a convenient representation for accumulating and sharing object-relevant information between processes. Sensory and categorization processes provide sensorimotor attributes for an object schema, and plan fragment processes govern and report on action outcomes relative to an object schema. Because they are all part of the same object schema, each process knows exactly where to look for data about its object schema that might influence its processing.

This bundling of processes in object schemas is not merely for convenience. Many models for planning actions towards objects represent objects separately from the processes and actions that act on them. While this separation does not preclude computation of plan likelihoods as in these example interactions, it does make it difficult to cleanly provide a foundation for phrases such as "the *liftable* object" or "the *graspable* object." This difficulty arises because the separate representation of objects, actions, and plans in typical models provides no single point of contact for these language elements. Making plan fragment processes uniformly part of object schemas makes it possible to represent affordances like "liftability" or "graspability" cleanly as part of the language-connected object representation, in addition to making such data available for plan evaluation.

1.1.5 Interaction 4: Claiming Data from Unbound Processes

Trisk is moving to lift a rectangular red block for exploration. As the hand is about to arrive at the block, the human moves the block into the path of the hand *such that the hand cannot complete its current motion. The hand bumps into the block, and Trisk notices the excessive force and moves the hand up, away from the collision. Trisk then moves the hand to an adjusted position, appropriate for grasping the block at its displaced location.*

The movement of the arm away from a collision is due to the third and final primary motivation in the current implementation, the *collision-avoidance motivation* (which was also depicted in Figure 1-4). Its role is to react to signs of collision by instantiating and supporting actions to move the arm away from a collision.

Collisions are detected by *unbound* behavior processes, which are processes that are not *bound to* (part of) any object schema. An unbound behavior process has its own dedicated interaction history in the belief context. When a collision-detection behavior process notices excessive force, it notates the position of the collision in its interaction history. The collision-avoidance motivation reads this collision data and supports a new unbound action process whose role is to take control of the arm and move it away from the collision.

Although written by an unbound process and therefore not associated directly with an object, the collision data can be treated as a sign of an object as well. The touch tracker of each object schema typically writes a location attribute when the hand is believed to be grasping the object, but it can also write a location attribute when unbound collisions fall within a preset distance of its designated object.

Thus, in the example, the location of the unbound collision is within a distance threshold of the expected location of an object, in this case the block being targeted for grasping. The touch tracker of the object's schema claims the collision data and writes it as a location attribute for its object schema. This new location data is used as the next grasping target for the hand instead of visual data because the hand's current position when attempting to grasp occludes visual contact with the object.

One of the other features of the model is the ability of the planning system to directly affect reactive processes. When the collision-detection processes detect collision-level forces in the fingertips, the hand is moved away from the point of collision. The exception is when the collision-level forces are encountered in the direction of an anticipated grasp. Part of the plan fragment for grasping an object involves disabling reactions to specific collisions that are expected as normal consequences of a successful grasp.

This example demonstrates the collision-avoidance motivation, as well as showing the reactive interactions between unbound processes, bound processes, and the planning system.

1.1.6 Interaction 5: Multimodal Discrepancies

Trisk is reaching to grasp and move a red ball. Just as the robot hand begins to close around the red ball, the human suddenly switches the red ball with a nearby green block. Initially, Trisk believes it has grasped the red ball, but as it begins to move its hand, it notices that the red ball is staying near the initial location. It immediately revises its belief of the red ball's location to the perceived visual location of the red ball, and creates a new object schema for the object in its hand, out of uncertainty about exactly what it has grasped.

When an object is grasped, the touch tracker provides a location attribute based on the current hand position. The model assumes that the object being grasped is the originally targeted object. However, visual tracking continues to provide a location for the object as well. A process called the *coordination process* is responsible for reconciling disparate attributes from multiple sources and writing a final attribute value to the object schema's interaction history. Each object schema has a coordination process upon creation, which stays coupled to the object schema.

Because both a visual and a touch location attribute are available for the red ball's object schema, the coordination process opts to go with the touch location as the reconciled location attribute. The system is designed this way because the touch location is a more precise, three-dimensional location value based on the hand's position in space, while the visual location (in the current implementation) is based on a single two-dimensional camera input. Furthermore, visual segmentation is highly prone to error from shadows, occlusions, and spurious regions in the area near the robot arm and hand.

As the robot begins to the move the object, though, the red ball's visual location fails to move, and the coordination process is increasingly willing to make use of the visual input as the hand moves away and the visual input becomes more reliable. At some point, the distance between the red ball's touch location and visual location constitutes a large enough discrepancy that the coordination process opts to detach the touch tracking process, allowing it to instantiate a new, unrelated object schema. The touch tracking process subsequently provides touch data for the new object schema. The coordination process for the original object schema then has only the visual input to use for location reconciliation.

The new object schema initially has no visual characteristics because segments coming from near the hand are untrustworthy. Only when the hand drops the object and moves away will the system evaluate the object's visual properties.

The point of this example is to describe the reconciliation of object attributes based on simultaneous multimodal inputs. It also shows the coordination process reconciling location data and detaching a process to maintain the coherence of the object schema's identity.

The maintenance of object identities is a key role of the object schemas. Visual input is subject to occlusion, shadows, noise, and spurious regions. Extracting coherent object identities from noisy visual input and unreliable touch input is important for resolving verbal references and carrying out plans.

1.2 Model Summary

In this section, I will summarize the structures in the OSM introduced via the examples. The description of the model can be divided into approximately four parts: behavior processes, the belief context (which includes the object schemas), the planning system (with plan hierarchies and the primary motivations), and the language system. Because object schemas and plan hierarchies are both composed of behavior processes, and language input leads to the creation of behavior processes, the parts of the model are fairly interdependent. The four parts are more like different perspectives in which behavior processes are organized and manipulated, rather than separate modules of processing that pass messages back and forth.

Behavior processes are like thread objects, each of which runs concurrently with the rest of the system. Each object schema is composed of behavior processes (which are "bound" to the object schema) and an interaction history. Each behavior process writes attributes and outcomes to the interaction history of its bound object schema, with the exception of unbound behavior processes, which write data to their dedicated interaction history. Each plan fragment process also writes its outcome data to an additional interaction history. All object schemas and interaction histories are stored in the belief context.

Each object schema also comes with a coordination process, which is responsible for reconciling multimodal attributes and resolving discrepancies by instantiating and detaching other object-bound processes.

Actions taken by the system are instantiated and supported by the three primary motivations: avoiding collisions, responding to verbal input, and exploring objects by lifting them. The primary motivations instantiate and support behavior processes, sometimes leading to the construction of a plan hierarchy consisting of condition processes and plan fragment processes. Leaf nodes in the plan hierarchy are action processes that inherit the support from the primary motivations via the plan hierarchy, and subsequently execute their motor actions. Conditions and plan fragments are also object-bound behavior processes, thus allowing rapid re-evaluation of the plan hierarchy amid changes in the external environment or internal information.

Language input comes in the form of parse trees, which have verb tokens that translate into plan fragments and noun phrase tokens that translate into reference processes. A reference process matches an object schema by comparing the object schema's attributes with the description tokens in the reference process. A plan fragment can be generated from the verb token in the parse tree, and leads to a plan hierarchy to guide action.

1.3 Key Contributions

The five example interactions are intended as a brief overview of the properties of the OSM. The main contribution of the thesis is the OSM, a model that permits integration between aspects of language grounding and aspects of behavior-based design. Within this integration, there are several secondary contributions:

- The identity and attributes of an object schema are reconciled and adjusted rapidly amid changes and noise in the external environment, while being stable enough for action planning and language use. The organization of behavior processes into object schemas provides structure for data sharing in order to maintain object identities.

- Language and direct perception both lead to object attributes, which can then influence action planning due to outcome expectations based on past experience. This gives attributes a role in the model beyond grounding words for reference resolution.
- The presence of behavior processes within plan hierarchies enables opportunistic replanning and use of new information, including verbally-sourced information, as soon as it is available.
- The presence of action-related behavior processes and outcome expectations within object schemas provides a single structure for grounding of terms related directly to object affordances and object-directed actions, such as "liftable" or "graspable."
- The state of the plan hierarchy can adjust the immediate responses of the reactive processes. For instance, certain collision forces from the finger touch sensors do not trigger collision responses when attempting to grasp an object.

1.4 Overview of Chapters

The purpose of the current chapter is to give an introduction to the motivations and contributions of the thesis, and a brief overview of the model and implementation. The subsequent chapters are arranged as follows:

- Chapter 2 describes the related works and influences that inspired this system.
- Chapter 3 presents the model in detail.
- Chapter 4 explores the mechanisms underlying the implemented system and explains one of the example interactions in greater detail.
- Chapter 5 presents several means of evaluating the model, including a discussion of language use made possible by the model. It also explores future directions and conceptual limitations of the model.
- Chapter 6 reiterates the contributions and the key points of the model.

Chapter 2

Background, Influences and Related Works

The main idea of the Object Schema Model is to construct object schemas that interact continuously at the sensor and motor level, while providing a stabilized yet responsive level of discrete symbols for language and planning. The basic ideas underlying this work derive from a number of other works spread across various fields. This chapter describes some of these influences and related works. First, I discuss the use of object abstractions and object schemas. Then I describe and compare some behavior-based and object-centered systems. Finally, I summarize language-related computational models.

2.1 Object-Level Abstraction in Humans

First, the question of why objects are a useful abstraction bears some discussion. Then, there is an additional question of what mechanisms really underlie the perception and abstraction of objects.

Objects are an abstraction that humans extract from perception in order to operate on the world. This can be explained in terms of what Brian Cantwell Smith [76] calls "flex-and-slop" – with respect to the survival needs of an organism, the world can be viewed and operated on with a certain amount of flexibility and still yield the same results. In this view, an object is a separable entity because the perceived boundaries between it and the rest of the continuous stuff around it are helpful to the purposes of the perceiving organism.

For example, if a creature's surivival can be aided by eating a piece of fruit, it makes little difference what angle the creature approaches the fruit from, or exactly what trajectory it uses to grasp and manipulate the fruit into its mouth, as long as it eats the fruit. The creature leverages the flex-and-slop in its environment by automatically ignoring distinctions that have no bearing on its survival. Discretizing the perceived world into objects enables cognitive processing to be focused on abstractions that aid survival, rather than the intractable task of acting in a completely continuous world.

Object-level abstraction has been demonstrated in studies of humans. Pylyshyn et al. [62] shows that humans are capable of visually tracking 4 or 5 objects simultaneously as they move around. This ability shows that humans perceive their visual world in terms of discrete objects. The work of Ballard et al. [5] to study and model human eye movements shows that humans not only view the world as discrete objects, but they take actions with respect to objects as well. Ballard's work also shows that humans maintain sensory contact with objects by repeatedly fixating task-relevant objects, suggesting that humans do not work with static internal models of their environment.

Another way to see that humans abstract objects from their perceptions is to examine the content of language. Many words in human language represent something abstracted and categorized from continuous sensorimotor experience. Every noun phrase is evidence of a discretization process that views the world as a set of separable objects or concepts, including the concrete physical objects which are the focus of this thesis.

This is not to preclude levels of abstraction besides objects, nor is it to preclude the usefulness of sensorimotor behaviors that require no abstraction at all. My intent is to argue that objects are a necessary level of abstraction for coherent behavior in the world, given the need of a creature to act on its world to survive. Furthermore, I argue that object representations need to stay in contact with their sensory referents, to whatever extent is possible. Humans cannot see everything at once; as Ballard shows, humans actually only notice a small amount of information, typically limited to the task at hand. Thus, some amount of internal modeling and state is necessary to handle multiple or occluded objects. However, acting on a completely abstracted static model would lead to poor responsiveness in a dynamic environment.

2.2 Object Schemas

The term “schema” is used in psychology to refer to the context-dependent categorization processes that govern a person’s interpretation and prediction of the environment. Rather than acting based on raw stimuli, people filter their perceptions based on their schemas in order to focus attention and guide action. Notably, Piaget [61] studied and described the ways in which schemas are acquired, used, and generalized. In his descriptions of early children, Piaget’s schemas comprise a context, an action, and a result. Objects, for instance, could be treated as contexts towards which a specific action will have a predicted result.

The term generalizes from this usage to describe any sort of script or frame (e.g., Minsky talks in terms of frames [54]) within which people organize their perceptions in order to interpret and predict. My use of the term “object schema” thus describes a representation of an object that emphasizes the processes by which people compile perceptions and actions into a set of predictions. The behavior processes in my model perform actions and observe inputs in a continuous space, but the interpretation of the continuous data is done with respect to discrete schemas, to guide predictions for future behavior processes.

Drescher [25, 26] built a system that operated in a discrete grid microworld with simple interactions akin to vision, touch, and grasping. The idea was to do learning of Piaget-style schemas from raw percepts and actions, by compiling multimodal predictions on interactions with objects in the grid. Under Drescher’s model, an object is a pattern of regularity, a context that repeatably brings up a set of predictions on action and perception schemas. His eventual (but currently unrealized) hope was to then use these regularities as schemas themselves to bootstrap a general learning process to build higher order schemas.

Roy [69] develops a theoretical framework for connecting language use to the physical world via sensorimotor schemas. This culminates in the object schema as a representation of objects that is built from components such as action schemas and property schemas. Each object schema is a network of actions that could be taken on an object, tied to effects that would result from each action. As actions are taken by the agent, the actions and their effects are encoded within the context of the object schema.

The contribution of this thesis is to explore interactions that are made possible by focusing specifically on the object schema level of description, by predicting and perceiving actions and outcomes within the framework of their relation to objects. The notions of Roy's action and property schemas and Piaget's sensorimotor skill schemas are folded into the behavior processes of my model, which are the primary components of an object schema. The implementation described in this thesis extends models of schemas by developing details necessary for making use of object schemas in a real system, and adds an emphasis on the process view for guiding continuous interactions.

2.2.1 Objects as Sets of Affordances

A similar way of looking at the idea of the object schema is that objects can be viewed as the sum of their affordances (i.e., actions that can be taken with respect to an object and what they enable for the agent [33]). A number of works take this viewpoint towards objects.

Notably, Stoytchev [77] built a robot system that discovers affordances of a set of objects by attempting to use each one as a tool and observing the results. Fitzpatrick [28, 29] used a robot to segment objects based on physical manipulation, a significant demonstration of *active perception*, and also accumulated affordance statistics on the result of poking at objects. Modayil and Kuipers [55] use a mobile robot to learn the results of motor babbling in terms of effects on objects in the robot's perceptual space.

Although most affordance-based models attempt no connection to language, some projects do add language to affordance-centered object representation. In particular, Gorniak [37] constructed a language-interaction system based in a video game that represents objects in terms of their affordances. Compared to the OSM, Gorniak's work built an elaborate plan-recognition and planning system which made use of object affordances, but by existing in a video game environment could leave out the intricacies of continuous perception, active acquisition of object attributes, and probabilistic representations of action outcomes. On the human side, Glenberg and Kaschak [34, 35, 45] have performed human studies showing that linguistic processing can be influenced by mental processing of situated actions or of object affordances.

Affordance-based objects and object schemas are very similar in their focus. The underlying idea is that objects can be treated as the sum of their affordances, and that an object exists primarily as a target for motivated action. This stands in contrast to most approaches to object conceptualization, which treat objects solely as compilations of attributes like color, shape, and weight. By requiring objects to be represented in terms of actions, even a seemingly static attribute like color becomes the result of an active visual tracking process that provides information about the object only because it successfully tracks the object.

My OSM is consistent with this view of objects, not just because it views attributes as the results of active processes, but additionally because it uses object attributes like weight and color for prediction of action outcome likelihoods, rather than just as signs of object identity and hooks for adjectives. I choose to use the term "object schemas" because the idea of object affordances primarily emphasizes motor actions taken towards objects, while Roy and Drescher's views of object schemas retain both the sense of objects-as-affordances in addition to the sense that object attributes are sensed as a result of active processes. The term "schema" also carries with it the general idea of processing on other levels in addition to objects, which retains a connection to longer-term goals of representation.

2.2.2 Making Semi-Stable Constructs From Sensory Input

A key aspect of the OSM is the continuous updating of semi-stable representations based on sensory input and action outcomes. Mobile robot systems often use this paradigm for map-making and navigation purposes. The various implementations of SLAM (Spatial Location and Mapping), e.g. [12, 21] among many others, are based on the idea of building a map based on incoming multimodal sensor information, and then using the map to formulate action, and then continuing to update the map.

While SLAM maps are represented as elements in continuous space, Kuiper's recent incarnations of the Spatial Semantic Hierarchy (SSH) [46, 47] make use of a hybrid model that connects the SLAM-generated local map of an area to various discretized models of location. At the discrete level, the SSH uses state-action-state schema representations to model transitions and make decisions. Compared to the OSM, continuous updating of map structures is in spirit a similar idea to continuous updating of object schemas, and the use of a discretized level of representation enables planning to achieve goals. The OSM's focus on object representations leads towards language use with respect to affordance-based manipulation planning, as opposed to navigation planning. I believe both are useful directions.

A number of other robot and simulation projects at the MIT Media Lab have also made use of updates from multiple inputs to provide a stabilized sense of object identities. Because these projects also make use of action selection systems, and some also include representations of complete belief contexts, I will discuss them at the end of the next section.

2.3 Behavior-Based Systems

Early robot works such as Shakey the Robot [58] made use of what eventually was termed the "sense-plan-act" pattern of execution. Sensor information was used to build a model, a plan was devised based on the model and the robot's goals, and then the plan was executed.

The STRIPS planning system used in Shakey laid the foundation for what is considered "traditional planning" in AI. Actions available to the system are expressed as operators with preconditions and effects. Goal states of the system are explicitly expressed, and the effects of the operators are used to "back-chain" a sequence of operators from the starting state to the goal state.

While robots like Shakey were good early prototypes of robots acting in human environments, the slow speed of the planning in such systems generally made them poor for acting amid sensor noise and a dynamic environment. Chapman, in particular, explored the combinatorial limits of planning systems [20] before deciding that planning was ill-suited for making decisions in a complex and uncertain world. He eventually did away with planning and worked instead on designing Pengi, a system that played a video game according to a set of reactive rules [1].

The idea of behavior-based robotics was developed around the same time period in response to the need for robots to be able to take action in the real world. The subsumption architecture [14, 15] proposes that robot behavior be modularized into separate layers, each with a specific goal and the ability to perceive and act towards that goal. Goals in such systems are typically not explicitly represented, but rather they are consequences of each module's mechanisms. Lower layers are deemed more important for the robot's real-time performance than upper layers, and thus when a lower layer demands control of the robot, it inhibits the upper layers. Significantly, the definition of the subsumption architecture includes avoiding internal state as much as possible, in order to allow the world to be its "own best model."

While robots based on subsumption architecture do exhibit excellent reactivity and interesting behaviors, the requirement to have almost no state and to do essentially no planning makes it difficult to scale the architecture towards more human-level abilities such as linguistic communication or acting on occluded objects towards a temporally-distant goal. Nonetheless, the point was successfully made that reactivity and layered architectures are important in real-world robots and that the single-loop sense-plan-act paradigm is insufficient for robust real-world behavior.

After the success of the subsumption architecture, many subsequent systems started from the foundation of behavior-based control and added planning-like capabilities in order to extend its abilities. Various methods to organize and select actions, beyond simply having one preempt the other, have been devised as well. These will be described in the next few sections.

2.3.1 Action Selection Systems

Maes [50], Tyrrell [78], and Blumberg [11], among others, built action selection systems that allowed the pursuit of multiple goals by using various mechanisms to select and intersperse actions. Rather than the complete inter-layer inhibition used in the subsumption architecture, goals could be selected for pursuit based on urgency and applicability. These systems generally acted and sensed in simulated domains of varying complexity.

Maes' system [50] made use of a spreading activation network in which goals and environmental state activated nodes in a non-hierarchical network. Nodes representing executable actions competed to trigger their actions based on their activation levels.

Tyrrell [78] showed that Maes' model had trouble persisting behaviors over time and argued against the lack of hierarchies, and suggested a hierarchical feedforward network in which all goal nodes can activate action nodes simultaneously, with persistence built-in in the form of inhibition based on timing and uncertainty.

Blumberg [11] argued against Tyrrell's simultaneous activations, and instead proposed a winner-take-all hierarchical system in which hierarchy nodes of the same level are mutually exclusive, but in which nodes can communicate recommendations to each other. Activity fatigue and inhibition levels interacted to produce persistence.

What makes these approaches interesting is that they are capable of producing actions in service of sustained, explicitly-represented goals, while not conducting planning in the traditional sense. Furthermore, rather than having one goal or behavior consistently dominate, they permit multiple behaviors to compete to take action according to urgency and relevance. The explicit representation of goals in action selection systems is necessary for the goals to influence which behaviors are activated, so behavior selection can be performed.

2.3.2 Three-Layer Systems

The "three-layer architectures" described by Gat [31] add more elaborate planning as the upper layer in a three-layer model. Gat's own ATLANTIS system [30] is a notable example of such a model. The lower layer typically consists of reactive, relatively stateless behaviors, and the middle layer typically consists of what Gat calls a "sequencer," whose role is to pick a behavior to be executed.

The middle layer of the ATLANTIS system resembles the action selection algorithms described above, in that it keeps an internal state and makes use of knowledge about its current goal to select an action to take next, but without doing search or prediction. The upper layer, called the "deliberative layer," performed path-planning and self-localization, which amounted to planning using exhaustive searches in the mobile robot's domain. The upper layer in Gat's model is only operated on demand, when called upon by the middle layer.

Bryson [17, 18] embraces the three-layer model, but makes use of more complex semiautonomous behaviors that keep state, and chooses between them using a carefully-designed hierarchical action selection mechanism. The deliberative planning in her system takes the form of the hierarchical action selection.

In a similar spirit to three-layer models, some systems for planning character actions in video games, such as that used in the video game F.E.A.R. [59, 60], make use of action selection for serving top-level motivations, with low-level processes governing specific animations and actions, and high-level navigation and planning available on demand. In this specific example, the enemy characters are capable of opportunistic planning, in which changes in the environment or outcomes of interactions can cause a rapid revision of the plan hierarchy.

2.3.3 Comparisons to Behavior-Based Models

Because my Object Schema Model incorporates planning as part of a reactive system, while selecting actions in service of explicitly-represented motivations, it resembles the subsumption, action-selection, and three-layer systems. By running behavior processes in parallel and having them compete to execute, the OSM shares the reactive properties of subsumption-based systems, but has no preset priority scheme, instead allowing the primary motivations to determine which behavior processes should dominate.

Relative to the action selection systems, the OSM adds a much more complex sense of multimodal object identity and a connection to language, as well as using a real-world robot. Relative to the three-layer systems, the OSM is essentially using its object schemas as a middle coordination layer in conjunction with action sequencing, with a lower layer of behavior processes and an upper layer of action planning. It additionally uses the results of reactive behaviors to update its object beliefs and its plans, and adds connections to language at all levels of the system.

Orkin's F.E.A.R. video game agents perform opportunistic planning based on results of past interactions with the environment, which is a key aspect of my thesis. In the game, the agents perform actions in order to overcome obstacles on the way to a navigation goal. A failure to overcome an obstacle, such as a failed attempt to kick down a door, causes the obstacle to be labeled impassable, at which point the navigation system will find and select a secondary route, for instance leading the agent to jump through a window instead. Because obstacles are either passable or impassable, the agents in F.E.A.R. will never try the same door again if they fail once. The OSM performs a similar sort of opportunistic planning, but represents expected outcomes probabilistically, and is also designed to represent objects as more than just obstacles for navigation (for instance, as results of visual and tactile processes as well). However, seeing objects as obstacles for navigation in addition to the current set of affordances would be useful if the OSM were to be ported to a mobile or simulated platform.

2.3.4 Behavior-Based Systems that Use Objects

Several works make use of behavior-based paradigms while also representing objects for interaction, producing a number of similarities with the Object Schema Model. This approach is found, for instance, in a series of projects developed in Bruce Blumberg's group and continued in Cynthia Breazeal's group at the MIT Media Lab, by Matt Berlin, Jesse Gray, Damian Isla, and numerous others.

The C4 platform [19, 44] was a cognitive architecture for simulated creatures. It was used for development in several simulated domains, and served as a foundation for exploring some fairly sophisticated aspects of perception, representation, and learning based on ethological models of animal behavior. The sensory inputs of the simulated creatures was filtered in order to approximate what a real creature in a given situation would have been able to sense, avoiding the problem of "cheating" with an omniscient perspective.

In conjunction with C4, a model of object persistence [43] was devised in which object locations were tracked and predicted using *probabilistic occupancy maps*, a gridlike discretization of the environment with probabilistic presence values for given objects. Observed objects could be placed in the map, and then when not directly perceived, the objects' locations could be diffused through the map in order to revise expectations of where it might subsequently be encountered. This enabled a sense of object persistence, in which multiple encounters with an object could be reconciled into a consistent notion of the object's identity and location.

Distance metrics were used to reconcile objects using color, shape, and location, also taking into account the object's perceived velocity to control the diffusion of expected location. Each object thus tracked was termed a *belief* object, and the use of distance metrics to match perceived objects with belief objects was termed *belief matching*. Given the belief objects, it was then possible for the virtual creature to take action towards the objects, such as looking towards them or searching for them.

Later work [13] further extended the ideas developed in the C4 platform with application to the Leonardo humanoid robot in Breazeal's group. The perception and belief system of C4 was extended to use gaze information to model a belief system and a perspective for the interacting human. This information is then used for learning tasks such as sequences of button presses [13], simulated toy block manipulation [9], and a false belief task with objects in boxes [38].

The key similarities between these works and my Object Schema Model involve the compilation of object beliefs and, in the later Leonardo projects, the use of a hierarchical plan representation.

Object Beliefs and Schemas

Conceptually, the key difference between the compiled object beliefs of the C4 model and the object schemas of the OSM is that the object schemas encode the processes responsible for providing attributes and outcomes as part of the object schema. While the C4 and related models produce updated object beliefs using data from multiple perceptual inputs, the processes in the C4 model that generate those percepts are decoupled from the object representations. Furthermore, action and prediction in the C4 model are also handled by separate processes, decoupled from the object representations. The OSM's schemas constitute a more direct representation of objects in terms of affordances and processes, in contrast with representing objects as sets of attributes that are primarily used for tracking object identity. I believe that the use of processes and affordances as the main component of object representations cuts closer to the heart of why animals represent objects to begin with – as targets of action that enable specific results.

At the implementation level, the approaches do bear a strong resemblance. In terms of object perception, both approaches essentially reconcile objects using multi-dimensional distance metrics. One difference is that the object-coupled processes in the OSM provide a mechanism by which perceptual processes can be contextually attuned to the current state of the target object. A visual tracker tied to a specific object can be instructed to selectively disregard color, or position, or to run more frequently (in the case of scarce computational resources, where not all trackers run at full frame rate) in order to adapt to occlusion or motion. Such object-specific process activity is possible under any model, but is made especially convenient by including the process as part of the object schema.

Beyond the object-perception level, the object schemas include behavior processes that control object-directed motor actions and plans. Outcomes of actions can thus be compiled into the same representation as attributes from perceptual processes. This provides a convenient representation for prediction of action outcomes based on both perceptual attributes and past action outcomes. I argue it is also an ideal representation for grounding a range of terms like "heavy" (which is an attribute) and "liftable" (which is an outcome prediction) in a uniform manner in one single representation, all able to subsequently affect object-directed action planning. In models with actions decoupled from objects, no single point of representation is available for connecting words that describe affordances.

The bundling of processes and outcome predictions also enables a certain level of robustness to sensor noise not needed by the C4 and Leonardo platforms, which function in especially clean sensory environments – C4 ran in simulation, and some Leonardo tasks run in simulation while others use special visual motion-capture markers. In the Trisk implementation, if a spurious visually-perceived object has consistent characteristics, and the attempts to explore and lift the object are consistently met with grasp failures, these failures will be recorded with respect to the object and its attributes. After a few tries, the spurious object, and objects with similar attributes, will be ignored for subsequent grasping actions, constituting a sort of knowledge that certain types of visual objects have no tactile extent. Unfortunately, real-world sensor noise rarely has such clean noise characteristics, but the premise is nonetheless a useful one, and does work for specific cases.

In general, the bundling of processes into object schemas provides a convenient formalism for data sharing and reconciliation. In the Trisk implementation, information about tactile grasping and collisions can be reconciled into the same framework as visual segments, mean-shift visual tracking, and action outcomes. When one source of data is contextually more trustworthy or appropriate than another, the state of emphasizing or ignoring data from a particular process is kept as part of that object's specific representation, rather than as a set of parameters to a global merging function.

Planning, Belief Contexts, and Learning

In other ways, the C4 and Leonardo platforms make use of structures and abilities that go well beyond the currently implemented Trisk system, any of which would be welcome additions to the OSM implementation.

The Leonardo platform and the OSM implementation both include an explicit representation of planning. The chief benefit of the object schema approach stems from the inclusion of plans and actions as processes within the object schemas. This provides a structure in which changes to perceived object attributes, via vision, tactile contact, or language, can rapidly alter the current plan hierarchy.

On the other hand, the planning system used in Leonardo's false belief task additionally recognizes plans being executed by other agents, and can then perform back-chaining to complete such plans. Furthermore, Leonardo can maintain separate belief contexts (sets of object beliefs and goal/plan structures) for itself and multiple humans. Finally, both the C4 and Leonardo systems make use of various types of reinforcement and imitative learning, none of which are explored in the Trisk implementation, and all of which would be immensely useful.

2.4 Grounded Language Systems

Language in a robot-like system had its first start with Winograd's SHRDLU system [80], in which Winograd hand-built both a purely symbolic representation of a blocks world and a natural language grammar for interaction. Commands and queries were accepted and carried out by the symbolic robot "simulator." However, the connection of a system to the real world is a significant part of the challenge that cannot be overlooked. The noisy and dynamic nature of the sensorimotor world not only presents a significant challenge in terms of stabilization for language use, but can also present opportunities to directly experience the results of actions for establishing additional constructs for connection to language. Later, Winograd and Flores [81] came to criticize symbolic AI for its isolation from the sensorimotor world and from the holistic contexts that grant language a sense of meaning.

In recent years, numerous projects have been undertaken to begin connecting language to the real world (see [68] or the special issue that includes [69] for a review of a number of these). Some of these are primarily concerned with connecting word labels to specific perceptual characteristics. For instance, Regier's work [63] deals with representations for labeling spatial relations between pairs of objects. Work at Bielefeld on the BIRON robot [6] can translate visual attributes such as color and shape of objects from a toy construction set into word descriptions. Deb Roy's Describer system [65] identified and produced descriptions of rectangles in a simple scene description task based on color, size, location, and shape attributes.

Relative to these works, the OSM makes the representation of objects much more dynamic relative to the noisy, real-world environment. Furthermore, beyond just connecting the attributes of objects to words, the OSM uses attributes as elements in making predictions of action outcomes, which addresses an assumption made in language description systems about why the attributes would have been useful to collect in the first place.

Some grounded language projects move beyond object features to deal with actions. Bailey's thesis work [3, 4] used Petri nets to model actions and action features, which was intended for use on a simulated robot arm but was not fully implemented. Narayanan [56] used the same action-related formalisms as a framework for understanding metaphorical language use in news stories. Siskind [74] extracted force dynamic relations from video clips of actions to ground verbs using a framework of logical relations. The OSM models actions with less granularity than Bailey and Narayanan, and is not geared for recognizing human-generated actions in video input. However, the OSM differs by focusing on how actions relate to objects in the robot's environment; features such as action attributes, recognition in video, and metaphorical language use would be welcome extensions.

Verbal communication in robots is often used to direct mobile robots. In works like [7, 48, 49, 75], language is used to give commands that include both actions and spatial referents that refer to map features. As described in the section on making semi-stable constructs above, mapmaking does bear a resemblance to updating of object models, so these potentially constitute a step in a similar direction to this thesis, although the focus on object schemas in my model takes it in a different direction in terms of types of language to be grounded and types of plans to be formed.

Other interesting work in grounded language observes language use in humans in order to learn language from basic features. Roy's CELL model [64, 67] observed object shapes in conjunction with mother-to-child utterances and searched for co-occurring patterns in the phoneme sequences in order to derive associations between speech segments and visual shape and color categories. The CELL model was also implemented on a simple robot. Yu and Ballard [82] takes a similar approach for studying human language development by collecting gaze information, input video, and hand movements alongside speech, in order to provide as much context as possible for extracting co-occurrence information from the phoneme string.

As with many grounded language systems, my implementation of the OSM makes use of a speech recognizer to extract words from speech input, rather than attempting to learn the most basic word forms from phoneme strings as in the Roy and Yu works. Learning from phoneme strings is an interesting challenge, but it is impractical to attempt to handle too many different levels of interaction at once.

2.5 The Ripley Robot System

Previous work on our group's Ripley robot system [40, 41, 70] has been a significant influence on the features desired in the Object Schema Model.

Ripley is a robotic manipulator arm (see Figure 2-1) with seven degrees of freedom, including an end effector that grasps using a five-fingered claw with one degree of freedom. Two cameras are mounted at the end of the arm, enabling the system to see objects and humans in its environment, but only when the arm is facing in the proper direction.

The Ripley system makes use of a continuously-updating three-dimensional model of the robot's environment [40, 70]. This "mental model" is an OpenGL representation of the

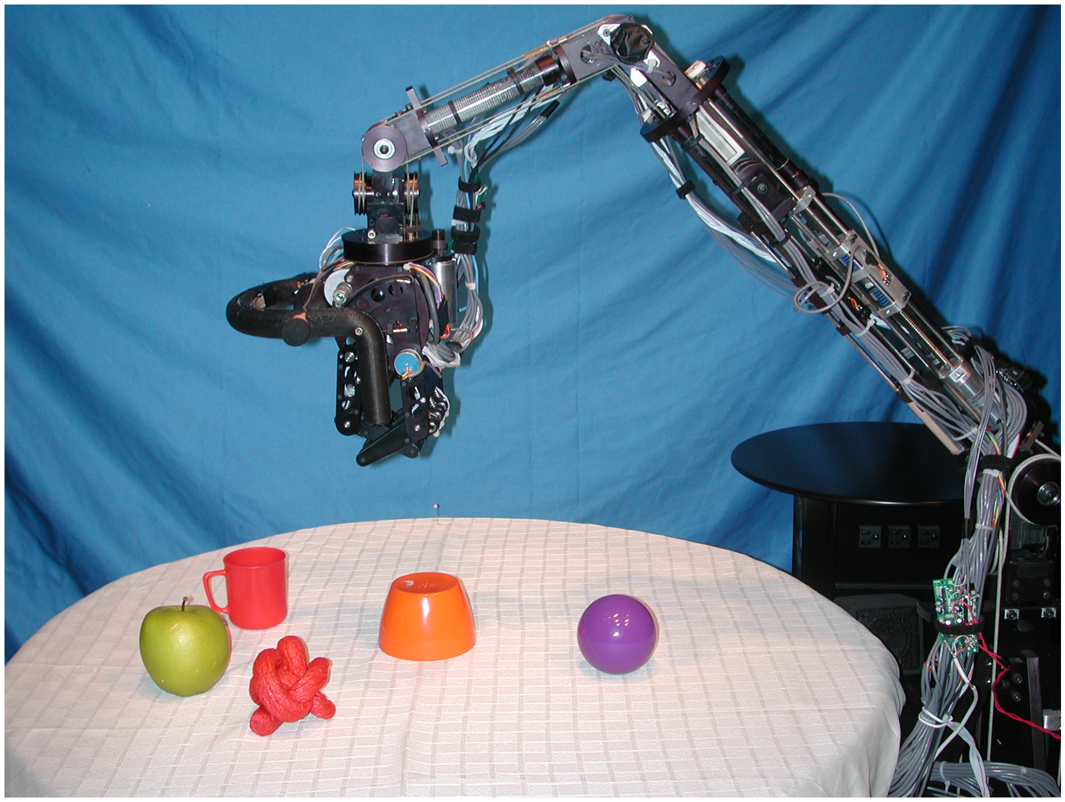

Figure 2-1: Ripley is a 7-DOF manipulator arm with a gripper claw and cameras at the end effector. It operates in a tabletop domain, seeing and moving small objects.

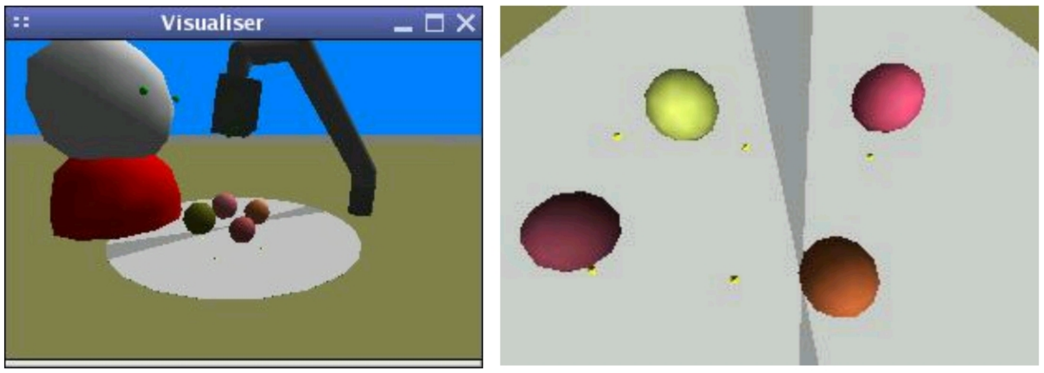

Figure 2-2: Ripley's mental model. Left: Ripley's visualization of itself, the human (green-eyed blob on the left), and four of the objects from Figure 2-1. Right: Ripley's simulated view from its own simulated perspective.

robot, the tabletop, objects on the table, and the position of the human (see Figure 2-2). The model is updated when objects or the human's face is perceived in the camera input and the robot is not moving. Objects are perceived using segmentation via a foreground-background color model, and the human face is perceived using the Viola-Jones face detector [79].

A speech recognizer [23] was used to take speech input, and keyword spotting enabled simple commands, such as "pick up" and "hand me" to be issued to the robot, with color terms used to pick out referent objects. Notably, the mental model enabled the grounding of referents such as "the blue one on my left," which when spoken by the human indicated the need to rotate the mental model's internal viewpoint to view the three-dimensional representation from the human's simulated visual perspective.

Finally, an action selection mechanism was layered on top of the mental model and the verbal command mechanism [41]. Three basic motivations competed for control of the system:

Heat motive The robot monitored its motor heat levels and used them to determine the urgency of this motivation. When in control, it led the robot to rest on the table.

Curiosity The robot was periodically directed to examine six visual zones in its environment. The urgency of this motivation was based on the amount of time since looking in that direction. When its urgency level dominated over the other motivations, it led the robot to look in the desired direction.

Verbal Interaction When commands were issued to the robot via speech, it would respond by acting on the command.

Figure 2-3 shows a visualization of the motivations competing to take action.

Curiosity interacted with the mental model because the mental model opportunistically updated itself based on visual inputs from whatever direction the robot was looking in. Likewise, commands issued to the system could make use of the information on objects and the human's position accumulated in the mental model.

However, Ripley's actions were taken using a single-loop sense-plan-act paradigm. Commands were carried out according to whatever the state of the mental model was when the

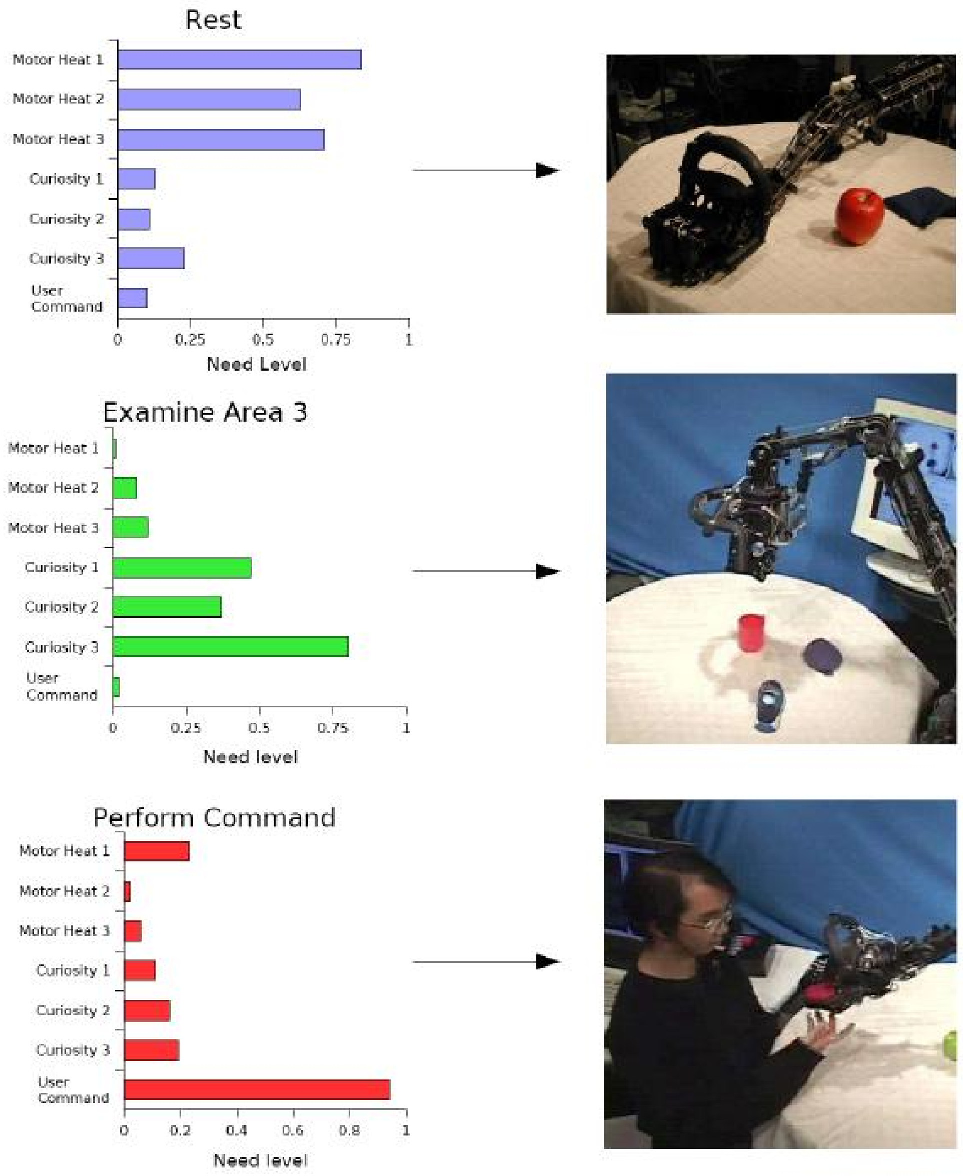

Figure 2-3: Depiction of Ripley's motivation system. Each set of motives has a priority, which allows it to compete for control of the robot. When motor heat is high, Ripley rests; when curiosity is high, Ripley looks around; and when the human issues commands, Ripley handles requests from the human.

command was received, and even if the mental model were to update while the robot was carrying out a command, no changes to the plan were possible.

2.5.1 Room for Improvement

Thus, one of the primary shortcomings of Ripley's sense-plan-act model was its inability to deal with failure in more than a hard-coded "try twice" way. If an object was moved by the human or displaced by the attempt to grasp the object, the world model could not update its expectations of either the robot's ability to grasp or whether there was actually an object there or not.

Furthermore, the Ripley system lacked any sort of real planning at all, merely carrying out hard-coded action sequences on demand. As such, no interruptibility was possible, so if the verbal interaction motive took control of the robot, it pre-empted the other motives until the human signaled the end of the interaction with a "thank you." Without a planner, the system was unable to keep state or plan in order to pick back up where it left off, so allowing the other motives to interrupt human-requested actions would only lead to behavioral incoherence.

Similarly, there was no interaction between the motivations and the behaviors they called upon. The completely non-interactive updating of the mental model meant that no physical inputs due to actions taken by the robot could alter its contents.

Thus, it was determined that the OSM would have to have a planner capable of allowing a behavior to be interrupted, to be continued later seamlessly from whatever state the robot ended up in. Also, the OSM would need a way for interactions throughout the system, physical and verbal, to alter the perceived properties of objects.

2.6 Summary

The goal of developing and implementing the OSM is to produce a single system that combines features from many of the described influences, with a model of process-based object schemas that enables novel forms of language grounding and coordination. Our previous experience with the Ripley system showed that combining various pieces of work together without designing for the integration from the beginning produces results that are good in some ways but incomplete in others. To recap the various types of influences and what the Object Schema Model adds:

Behavior-based systems clearly have an advantage in terms of acting in the real world, and the three-layer systems add the ability to perform action selection in service of explicitly-specified motivations. The object-centered behavior-based systems track and take action towards objects, but organize their actions differently from the OSM and thus provide inconvenient representations for language about object-directed actions or affordances. In general, the behavior-based systems are not designed to handle language, and many of them make simplistic assumptions about object perception.

Grounded language systems connect words to real-world referents, but mostly act on a static model of the world in a sense-plan-act manner, where the closest they come to “action” is to produce verbal descriptions of the world, rather than physically affecting it. Attributes that connect to language in such systems are used only for their connection to language, and not for planning and action. Works in grounded language in mapping and localization domains keep a dynamically updated representation of their environment, but, without an expressive representation of objects relative to actions, are mostly focused on descriptions of navigation actions and tasks.

Finally, the use of object-level abstractions in the OSM provides a level on which to coordinate the planning, language, and behavior systems. It acts as a means to collate information coming from the senses, as well as results (hence affordances) of actions, which in turn allows planning, language, and behavior to influence each other rapidly and usefully. Other key works that work with object schemas and object affordances address more specific aspects, or exist only as general models, without the connection to real-world robotic action, language, planning, and sensing.

Chapter 3

A Process-Centered Model of Embodied Object Schemas

This chapter defines the Object Schema Model in more detail; the mechanisms underlying the implementation on the Trisk robot with the actual sensors are reserved for the next chapter, although examples from the implementation will be used in the model description where necessary to clarify concepts.

As mentioned, the model can be described in four parts, each one a perspective that organizes and operates on the behavior processes in particular ways. The central component is the set of object schemas in the belief context perspective, which form semi-stable representations that stay connected to action and perception while providing stability for planning and language use.

See Figure 3-1 for a depiction of the four perspectives. Perspective 1 is the view of unorganized behavior processes and their execution. Perspective 2 is the belief context, which includes schemas and interaction histories. Perspective 3 is the planning system, which includes primary motivations, resources, and plan hierarchies. Finally, Perspective 4 is the language system, which creates reference processes for noun phrases and plan fragments for verbs.

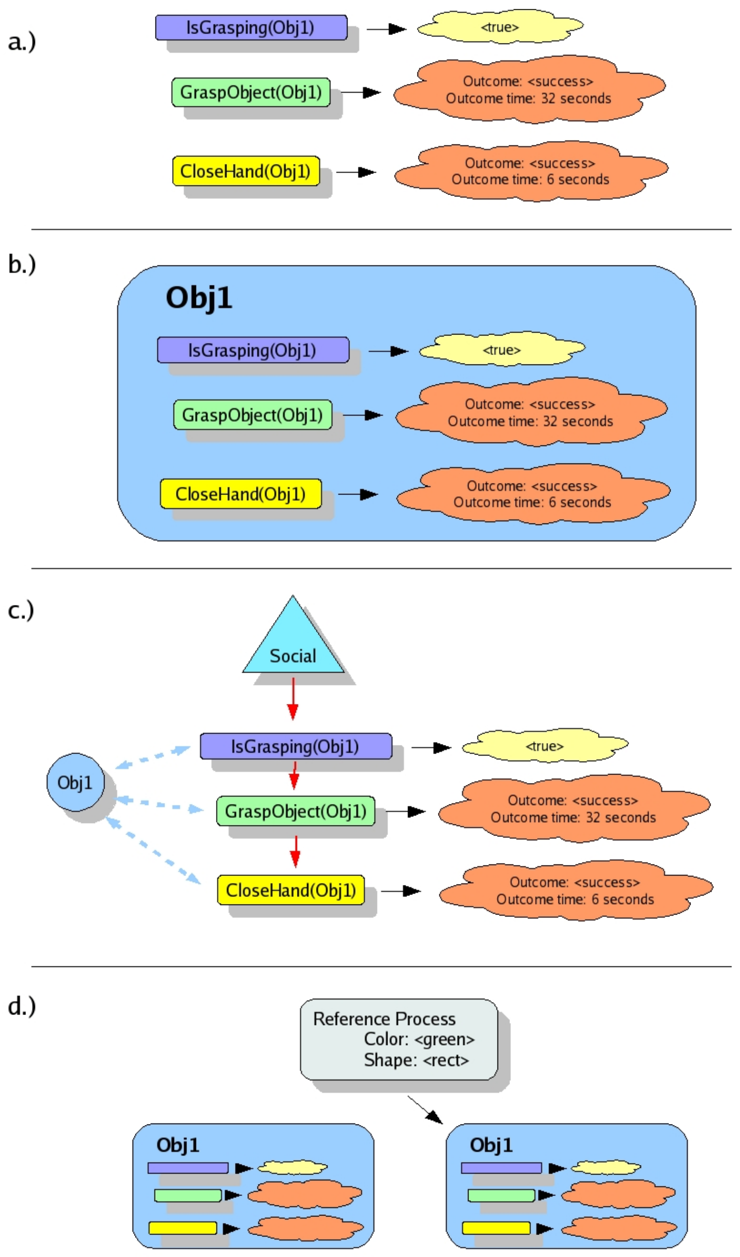

Figure 3-1: Illustration of the four perspectives of behavior processes. a.) Behavior processes viewed and running separately. b.) Behavior processes organized into an object schema. The condition "IsGrasping," the plan fragment "GraspObject," and the action process "CloseHand" all act upon and produce data pertinent to their object schema, Obj1. c.) The same behavior processes, organized into a plan hierarchy for grasping an object. Changes in the environment are rapidly reflected in the plan hierarchy, leading the planner to alter the hierarchy. d.) The language perspective uses reference processes, which themselves are behavior processes, to represent noun phrases. The reference process attempts to select a referent from the available object schemas.

3.1 Schema Basics

Before explaining the four perspectives individually, a few introductory comments about schemas will help frame the discussion. The object schema serves as the system's representation of an object in the environment. It comprises a set of active, instantiated, and potential behavior processes. Each behavior process in an object schema represents a process that is, or could be, executed with respect to the referent object.

Although examples up to this point talk only about object schemas, there are two other types of schemas in the belief context aside from object schemas: body schemas and location schemas, representing parts of the robot's body and locations in the robot's environment, respectively. They have the same structure as object schemas, but use a different set of behavior processes to perform their functions. Details of the other two types of schemas will be described in Section 3.3.3.

3.2 Perspective 1: Behavior Processes

The behavior process perspective is the view of unorganized behavior processes. Behavior processes (or just "processes," in this context) are like thread objects in a multithreaded program, which run concurrently and can be started and stopped at will. The functions of each behavior process may include reading from the belief context, monitoring sensor inputs, taking motor actions, and writing data back to the belief context. Some examples:

1. A visual tracking process uses past visual information about the object to make a prediction about the object's new location. It attempts to identify the object in each new visual frame and then records the object's perceived visual location in the interaction history.

2. A grasping process uses a visually-based prediction of the physical location of the object to move the arm towards the object. It monitors input from the fingertips to determine whether the grasp is successful or not, and writes outcome information (success or failure) to the interaction history.

3. A collision-response process monitors various physical attributes of the robot system

to determine whether the arm has collided with something. It writes collision data to its interaction history, which is then used by the collision motivation to reactively move the robot arm away from collision points. Collision data can also provide additional information for the current set of object schemas.

A behavior process is capable of real-world reactivity. In the examples given, the visual tracker is responding to motion of the real object; the grasping process can determine the success of the grip and retry accordingly; and the collision-response process reacts immediately to perceived collisions. Not all behavior processes are reactive in the current implementation, but this is a matter of implementation complexity rather than design; making behaviors more reactive leads to a generally more responsive system.

3.2.1 Behavior Classes and Binding Slots

Each behavior process is an instance of a behavior class. Just as behavior processes are like thread objects, a behavior class is like a thread class. Each behavior class can be instantiated into specific behavior processes (just like class instances), which can then be independently run. Behavior classes are part of a class hierarchy, with the following top-level behavior classes (see Figure 3-2):

- Sensory processes handle sensor inputs.
- Action processes (sometimes just called "actions" in the proper context) output motor actions and write outcomes to their interaction histories.
- Plan fragment processes (sometimes just called "plan fragments") coordinate sequences of action and condition processes.
- Condition processes (sometimes just called "conditions" in the proper context) return true or false depending on the state of something in the belief context.
- Translation processes read values from interaction histories and turn them into different values to be written back to the interaction histories.
- Coordination processes are created with a schema and are permanently bound to the schema. They coordinate other processes and reconcile data related to their schemas.

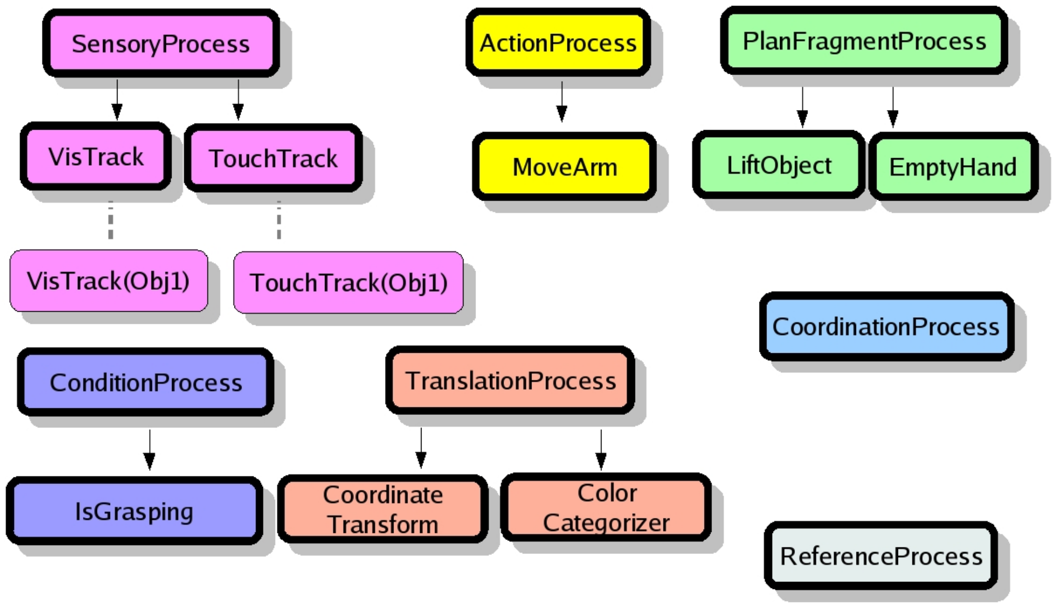

Figure 3-2: Part of the class hierarchy of behavior classes. Each top-level class has subclasses with specific, defined functions. Behavior processes are created when bound to schemas and instantiated from their subclass (as depicted for VisTracker and TouchTracker).

- Reference processes represent noun phrases for the language system and seek out object schemas with matching attributes.

As depicted in Figure 3-2, each top-level behavior class has various subclasses, each representing a type of behavior process that handles a specific function. Under the sensory process top-level class, there are VisTrack and TouchTrack subclasses that handle visual tracking and touch tracking. Just as with thread classes, each subclass is defined with a main loop function (*behavior function*) that will be called when the behavior process is activated.

A behavior process can thus be described as *instantiated*, which makes it like a thread object that has been instantiated (possibly with constructor arguments), or *active*, which makes it like a thread object whose main loop has started executing.

A behavior subclass has a preset number of *binding slots*, each of which defines a parameterized aspect of the behavior. The instantiation of each behavior process includes passing (*binding*) a schema to each of these binding slots. This is like passing parameters to a class constructor when instantiating a thread object. Note that the bindings can be changed after instantiation, if the process needs to be rebound to a new schema (due to detachment by a coordination process).

Instantiating a behavior process involves creating an instance of its class, and specifying schemas for each binding slot. Just like passing arguments to a constructor function, these are specified as a mapping that provides schemas from the belief context as values for the parameters of a behavior subclass: `(a : Obj1, b : Loc1)`, where `Obj1` and `Loc1` are schemas. When the behavior process is later activated, the mapping will be used to call its subclass' behavior function: `f(a = Obj1, b = Loc1)`.

3.2.2 Binding to Schemas

When a behavior process is instantiated and bound to schemas, that behavior process is considered a part of each bound schema. A process with multiple binding slots can thus be part of multiple schemas, and the function of the process will be different to each of its bound schemas. Not all behavior processes are bound to schemas. For instance, the unbound collision-detection processes act according to parameters of the physical robot, rather than parameters of object schemas. As such, collision-detection processes have no binding slots (i.e., function arguments).

Binding of a behavior process to an object schema generally means that the action of the behavior process will be targeted towards the object represented by that schema, and sensations monitored by the behavior process will be recorded in the interaction history of its bound object schema. In the case of behavior processes with multiple binding slots, each bound schema is treated differently according to the function that implements its main loop.

Describing a behavior process as bound to a schema and as part of a schema amount to the same thing, and are treated equivalently in the model. Describing behavior processes in terms of function bindings (as in the bottom half of Figure 3-3) is useful for seeing the system in terms of process-like functions, and describing behavior processes as parts of object schemas (as in the top half of Figure 3-3) is useful for seeing the system as representing a set of objects, each the sum of behaviors that can be taken towards them.

3.2.3 Instantiating Behavior Processes and Resource Competition

Let $\mathcal{M}$ be a mapping from binding slots to bound schemas, $\mathcal{M} : \mathcal{B}_f \mapsto \mathcal{S}$. It maps from the binding slots $\mathcal{B}_f$ of a behavior subclass to some schemas $s_0, s_1, ...$ from the set of all belief context schemas $\mathcal{S}$. The subscripted f in $\mathcal{B}_f$ is the behavior function that defines the behavior subclass. In the given example, $\mathcal{M}$ is the mapping `(a : Obj1, b : Loc1)`, and it maps from the binding slots $\mathcal{B}_f =$ `(a, b)` to the schema list `(Obj1, Loc1)`.

A *potential behavior process* is thus represented as a tuple $B = (f, \mathcal{M})$, by specifying the behavior function and a mapping of bindings. Potential behavior processes are not instantiated yet, but are a representation of a possible instantiation. The list of potential behavior processes is generated by permuting the set of behavior types with the set of all current schemas. This list is used by the planner to make suggestions for conditions. For instance, in order to satisfy a condition like `Grouped(Obj1, Obj2)`, the planner might try all combinations of `Move(x, y)` with current schemas. Each combination of a behavior subclass with a schema is a potential behavior process, which could then be instantiated if it seems suitable for the condition.

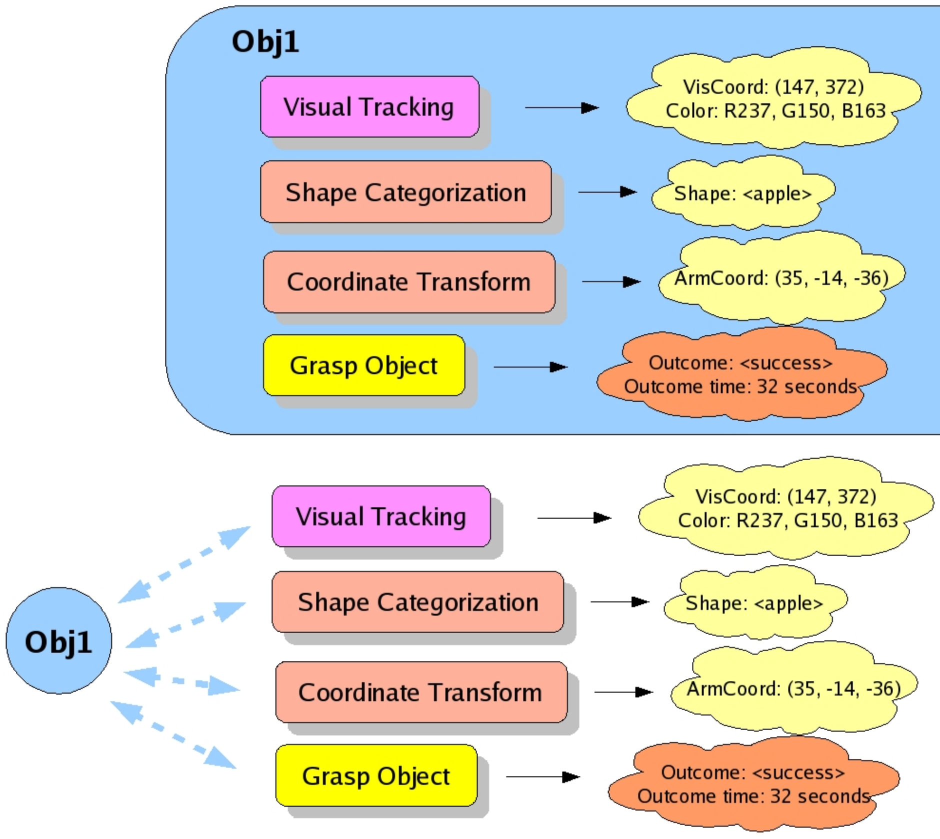

Figure 3-3: Top: Behavior processes are viewed as components of schemas. Each behavior process in a schema is a behavior that can be taken with respect to that schema. Bottom: Behavior processes are equivalently viewed as bound to schemas, as if the schema were a parameter passed to the function that defines the process. The dotted blue arrows graphically depict the connection from a schema "token" to the behavior processes bound to it. Viewing the schema as a token to be passed around the system, rather than as a container of numerous behavior processes, is useful for explaining the model in terms of objects, classes, and function calls.

Behavior processes are instantiated by currently active processes, the primary motivations, and the planner. For instance, a tracking process can be instantiated when a sensory process notices an unexpected visual or touch contact. An action process can be instantiated when a plan fragment process requires it at a sequence point. The primary motivation for curiosity creates a plan fragment process to explore an object whenever a new object is encountered. The planner creates new plan fragments and condition processes in building its plan hierarchy.

Behavior processes that are selected for instantiation must compete in order to become active and run their behavior function. Each behavior subclass definition includes the set of *resources* needed for execution. For example, the behavior function may need to control motors, or the planning system, or a dedicated visual tracker. Resources are controlled by the planning system, and they dictate the activation of behavior processes.

Once a behavior process is instantiated according to the data given by its mapping $\mathcal{M}$, it competes for the resources listed in its subclass definition according to *priority scores* in the range $[0, 1]$. The priority score represents the level of support stemming from the primary motivations, and is propagated between behavior processes according to the structures in the planning system. Priority scores are hand-coded, selected to result in behavioral coherence, and kept between 0 and 1 for ease of debugging.

An active behavior process is an instantiated behavior process that either has no necessary resources, or has the highest priority score of all processes competing for each of its requested resources. When this is the case, it begins executing its behavior function f with the set of bound schemas defined in its corresponding $\mathcal{M}$. Behavior functions are designed to loop as necessary, with each loop cycle running quickly and keeping simple state. During each cycle, a behavior function can transmit commands to motors, read the state of robot parts or elements of the belief context, and write to interaction histories.

3.2.4 Summary

To summarize, Perspective 1 contains the unorganized behavior processes. Behavior subclasses are defined by their behavior function and a set of resources, and can be instantiated into behavior processes by providing schemas for the binding slots. A potential behavior process is a behavior subclass paired with a list of schema bindings for its slots. An instantiated behavior process is competing for its resources according to a priority score provided by the planning system. An active behavior process has won its required resources, or has no resources, and so its behavior function begins executing. Processes can be instantiated by various elements of the system, but in most cases require additional support from the primary motivations in the form of priority scores to compete for activation.

3.3 Perspective 2: Schemas and The Belief Context

The belief context contains schemas and interaction histories. The behavior processes discussed in Perspective 1 are organized into schemas (except for unbound processes), and then each schema, unbound process, and plan fragment subclass gets assigned an interaction history. The interaction histories are implemented as a global shared memory space that enables behavior processes to communicate data and messages to each other. Information in the interaction histories is used to compute affordances for objects, and is supplied to the planning system so it can decide on courses of action. A diagram of the sorts of interaction histories found in the belief context can be found in Figure 3-4.

3.3.1 Schemas and Histories

A schema is a container within the belief context that holds a set of behavior processes. The behavior processes are bound (by filling binding slots) to the schemas that contain them. The schema is linked to an interaction history, which records properties and messages from the schema's constituent processes.

More formally: the belief context is a tuple $\mathcal{C} = (\mathcal{S}, \mathcal{B}_u, \mathcal{H}_s, \mathcal{H}_u, \mathcal{H}_p)$, where $\mathcal{S} = \{s_0, s_1, ...\}$ is the set of all schemas, $\mathcal{B}_u = \{b_{u0}, b_{u1}, ...\}$ is the set of unbound processes, and $\mathcal{H}_s = \{h_{s0}, h_{s1}, ...\}$, $\mathcal{H}_u = \{h_{u0}, h_{u1}, ...\}$, and $\mathcal{H}_p = \{h_{p0}, h_{p1}, ...\}$ are the set of all interaction histories of schemas, unbound processes, and plan fragments, respectively, indexed by schema, process, or plan fragment subclass.

Each schema is a set of bound behavior processes $s_n = \{b_0, b_1, ...\}$. The constituent processes b_i are each an instance of a behavior subclass, given as $(f_i, \mathcal{M}_i)$, as described in

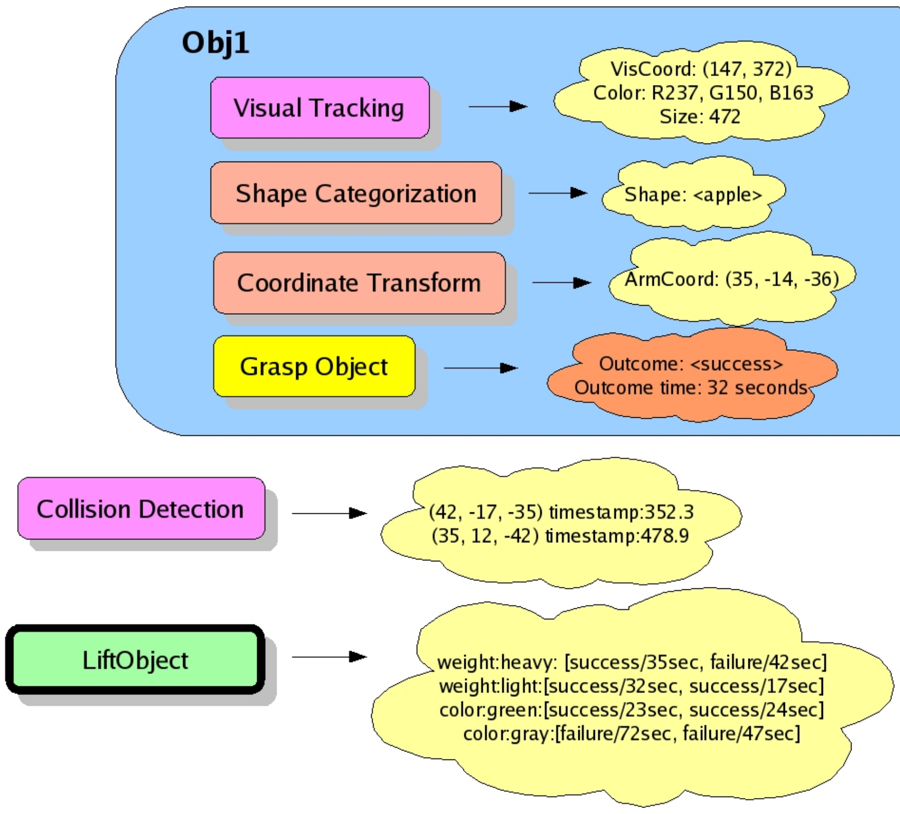

Figure 3-4: Depiction of three types of interaction histories in the belief context. Top: An object schema's behavior processes write attributes and outcomes to its interaction history. Middle: Unbound behavior processes, such as collision-detection processes, write to a separate interaction history that can be read by bound processes and primary motivations. Bottom: Each plan fragment subclass records an interaction history of outcomes resulting from each instantiated process. Outcome data can be used to determine success rates for future attempts relative to object attributes.

the section on behavior processes above, with behavior function f_i defining the function to be executed for that process and $\mathcal{M}_i$ defining the mapping of binding slots to bound schemas for that process. When active, each behavior process can execute its function f_i, which reads a programmed set of inputs from the belief context and writes a programmed set of outputs to the interaction history h_j corresponding to the bound schema s_j in the binding mapping $\mathcal{M}_i$.

Interaction Histories

The interaction histories h_j are implemented as a global shared memory space, in which all entries are indexed according to schema, unbound process, or plan fragment subclass. The interaction history h_j comprises the set of entries indexed under its associated schema s_j. Likewise, the interaction history h_k comprises the entries indexed under unbound process b_k, and the interaction history h_P comprises the entries indexed under plan fragment subclass P. A bound behavior process b_i can write only to the history corresponding to its bound schemas, given in $\mathcal{M}_i$.

The data written to the interaction history of an object schema can include continuous data, such as RGB color or size produced by tracking processes, as well as discrete data such as categorized color and shape and action-related data such as successes and failures of actions. Because all such data is produced as a result of an active process targeting the object, the data constitutes a history of the system's interactions with that object, hence the name.

Because all interaction history data throughout the system can be read by any process, the interaction history also serves as the primary communication method between behavior processes, allowing properties to be shared and messages to be passed. Organizing the data by schema and process makes it easy to determine the subject of each piece of information.

Attributes and Outcomes

Some data written to an interaction history is written to a specific labeled field attached to a schema, called an *attribute field.* Examples of attribute fields might include RGB continuous color, discrete color categories, discrete shape, and behavior success information. The data written to each attribute field is called a *attribute value*. In these examples, the attribute values might be (232R, 171G, 83B), `red`, `ball`, and `true`, respectively. Together, a pair like `color:red` is an *attribute*.

Other data written to an interaction history involves an outcome or a prediction with respect to an action or plan fragment. For instance, an instantiated plan fragment process to grasp an object will have a corresponding success expectation written to the interaction history. When activated, the plan fragment process then writes additional information about its running state, and eventually writes a success or failure outcome to the schema's interaction history.

Interaction histories are indexed by schema for reading and writing of data, but when successes and failures of plan fragment behavior processes are added to an interaction history, these results are also stored, along with attributes of schemas bound to the plan fragment process, indexed by plan fragment subclass. Indexed like this, it is then possible to use this information to make the predictions about outcomes of object-directed actions based on object attributes. This will be described in more detail in Section 3.4.4, on the prediction of success rates in the planning system.

New Schema Creation

Any behavior process can request the creation of a new schema in the belief context. This typically occurs as a result of a sensory process noticing signs of a new object, either by seeing an unexpected visual region or by noticing an unexpected touch contact. Once the new empty schema is created, it is typically populated immediately with a new tracking process that monitors sensory inputs for continuing signs of the object, and a coordination process, described in the next section. Additional behavior processes are instantiated and connected to the schema based on requests from the planning system and the coordination process.

3.3.2 Coordination Processes and Schema Management

Each object schema is automatically bound to a *coordination process*, which is a behavior process permanently linked to the object schema. The coordination process manages the other processes pertinent to its schema, and also manages information in the schema's interaction history. Some examples of actions taken by the coordination process in an object schema:

1. The coordination process can spawn shape and color categorization processes to turn visual inputs into category attributes for linguistic labeling and plan prediction. The output of categorization processes is written to the interaction history as semi-stable object attributes for use in planning and speech interaction. Such object attributes are only semi-stable because they can be altered based on subsequent information from the interacting processes.

2. If the visual tracking process has a poor confidence, the coordination process can deactivate it. The coordination process then begins checking incoming visual data in the interaction histories to reestablish visual contact with its object and restart the visual tracking process.

3. As mentioned in Interaction 5 in Chapter 1, when multiple behavior processes are interacting with the object, the coordination process checks for disparities in incoming data via the interaction history to determine whether the object is still a single coherent entity or if, for instance, the robot may have targeted one visual object but accidentally grasped a different one. In this example, the object will be initially recorded as being grasped, but the visual tracker will report no motion of the tracked object, even when the grasping process starts reporting motion. This disparity will lead to decoupling of one or more of the behavior processes from the object schema, allowing the newly unbound processes to spawn a new object schema.

As seen in this last example, the activity of the coordination process creates a dynamic coupling between behaviors in multiple modalities for each object schema. Visual observations lead to predicted locations in the physical domain, and vice versa, and discrepancies in predictions lead to reshuffling of behavior processes that can create and modify object schemas.

3.3.3 Types of Schemas

Aside from object schemas, there are two other types of schema in the belief context: location schemas, and body schemas. Location schemas represent locations, such as "above the table," or "to the left of the blue ball," and body schemas represent data about the robot's body, such as arm and head positions, motor heat levels, and collision information. This allows robot parts and locations to be bound to the binding slots of behavior processes in the same way as object schemas.

The key difference is that location and body schemas consist of a different, and less diverse, set of processes. Locations, unlike objects, are not visually trackable nor physically collidable. This means that rather than having a set of processes reporting attributes, a location schema only has its one process, its coordination process, to manage and evaluate its location value. Likewise, an body schema has only two processes—the coordination process, to write incoming embodiment data to the interaction history, and a translation process to perform coordinate transformation between visual and physical coordinate systems.

Thus, the three main types of schema—object, location, and body—are primarily distinguished by the functionality of the coordination processes assigned to the schema based on type.

Within the three main types of schema, it is possible to further distinguish between subtypes. For instance, the schema representing "the left of the red ball" is a location schema, but is also an instance of a subtype of location schema, "the left of `x`." The subtype makes use of a more specific coordination process whose role is to compute a point to the left of the location of a *landmark object*, in this example a red ball.

Similarly, the body schemas representing arm parts differ from the body schemas representing elements of the robot's hand. The coordination processes of each subtype of body schema connect in different ways to the network to acquire their body information.

In contrast, object schemas are all instances of the most general object schema type. This means that the coordination processes of object schemas share the same functionality, and that the constituent processes of object schemas are the same. However, the content of the interaction histories is used to vary the way the system treats each object. For instance, an object with a low likelihood of successfully being grasped essentially amounts to an “ungraspable” object. Thus, subtypes of object schemas derive from their attributes and outcome expectations, rather than intrinsically from their coordination processes.

While object schemas and location schemas are different in nature, both types of interaction history data have to deal with coordinates. As such, both schema types do share several type of processes, such as coordinate transformation processes. The processes involved in object schemas can be seen as a superset of the processes involved in location schemas.

3.4 Perspective 3: Planning

As described earlier, the behavior processes execute their behavior functions if they can win resources based on their priority scores. The origin of these priority scores is the planning system. Behavior processes with no resource requirements, such as simple visual trackers and body processes that read values from the network, can run without intervention by the planning system, but many behavior processes need control of the arm, hand, neck, planner, or secondary visual trackers. To successfully compete for these resources, the processes need priority scores from the planning system.

Briefly, the planning system works as follows: The primary motivations produce priority scores for action processes, condition processes, and plan fragment processes. Based on these priority scores, the planner uses backchaining to produce a plan hierarchy, which allows the priority scores to propagate from the behavior process at the root of the hierarchy to each of the leaf nodes in the hierarchy. The leaf nodes represent actions that are expected to lead to the completion of the behavior at the root node.

Like schemas, the plan hierarchies are structures that contain and organize behavior processes. Based on the current state of a plan hierarchy, new behavior processes are instantiated and used to further grow the hierarchy.

The primary motivations, plan hierarchies, priority scores, and resource competition together form a *behavior selection system.* The basic structure of the planning system was depicted back in Chapter 1, in Figure 1-4.

3.4.1 Primary Motivations

At the root of the planning system is the set of *primary motivations.* Primary motivations examine the belief context and decide to instantiate and support behavior processes that are expected to address the motivation's needs. The premise behind these motivations is that all actions by a live animal are taken in order to sustain its homeostatic equilibrium, and that by adjusting priorities, competing behaviors are interspersed, thus leading to the survival of the animal. Each primary motivation is a programmed routine that periodically suggests (by instantiating) and evaluates potential behavior processes (by passing priority scores). Priority scores used throughout the current implementation are chosen and hand-coded to ensure coherence in the robot's behavior. The primary motivations in the current implementation are:

- To avoid and move away from collisions ("collision motivation"). Collision-detecting processes write collision data to their interaction histories, and this motivation reads the data, creates collision-avoidance action processes, and supports them with a very high priority score.

- To explore objects in the environment ("curiosity motivation"). Currently, this means attempting to grasp and lift objects to explore their graspability and liftability. This motivation receives a very low priority score.

- To address requests by the human user ("social motivation"). Processes are suggested based on verbal inputs. This motivation receives a moderate priority score.

The motivations and action selection system of an animal play a major role in determining its evolutionary fitness. Likewise, the motivations for a robot or computer system should be selected to make it suitable for its purpose, which typically is to be helpful to humans. Being helpful to humans requires acting appropriately, as well as not destroying itself. Just as an animal with poor fitness would die off and make way for other genetic lines, a machine with poor fitness is bound to be replaced, or at least reprogrammed.

3.4.2 Priority Scores and Resources

Multiple behavior processes cannot concurrently use a limited resource. In the current implementation, the resources that processes must compete for are motor control, the planner, and the mean-shift visual tracker. As mentioned above, the competition for resources is governed by the priority scores assigned to each process, and only a process that wins its necessary resources can become active and run its behavior function. A small priority bonus is given to any process that is currently winning a resource, to prevent unproductive switching back and forth between processes (dithering). The primary motivations pass priority scores to behavior processes via two main pathways:

Direct Each motivation instantiates and directly assigns priority scores to specific behaviors, in certain states of the belief context. **Example**: When a new object is detected, the curiosity motivation instantiates and prioritizes a behavior process to grasp the object to explore its affordances.

Deliberative The primary motivation can instantiate a condition process or plan fragment process, rather than an action process. This process is then supported with a priority score. The highest such priority score wins the planner resource, and triggers the generation of a plan hierarchy. Priority scores are propagated through the hierarchy, thus allowing indirect assignment to appropriate actions. **Example**: The system is told to move an object to the left, so the condition that the object be at the target location to the left is created. The planner performs back-chaining to determine that the way to satisfy the condition is to reach, grasp, and then move the object, and constructs a hierarchy accordingly to enable each action to occur in sequence.

The two pathways are depicted in Figure 3-5. Only the deliberative pathway actually involves plan hierarchies. The direct pathway involves only the primary motivations and the relevant behavior processes, resulting in much faster reactivity for simple response behaviors like collision avoidance.

The behavior types to be supported by each primary motivation, and the priority scores, are currently hand-coded and adjusted to produce coherent responses. It should be possible to use reinforcement learning techniques to determine which behaviors lead to satisfaction

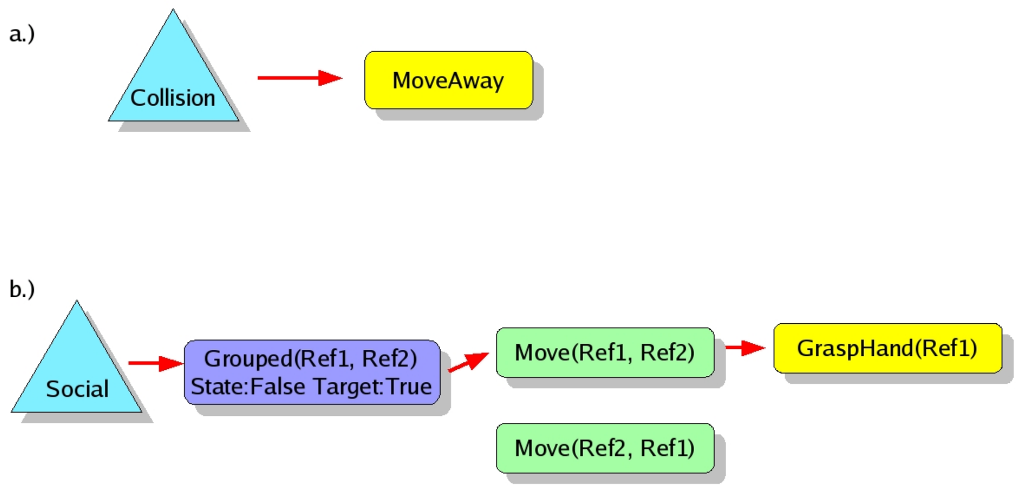

Figure 3-5: There are two ways that priority scores reach behavior processes. a.) In the direct case, the primary motivation passes the priority score directly to a process, which can then immediately activate. b.) In the deliberative case, the primary motivation passes the priority score to a plan fragment or a condition, and a plan hierarchy is constructed that allows the priority score to be passed from process to process until it reaches action processes that coordinate motor commands.

of each primary motivation; this would require an explicit reward system and a way to assign credit for rewards.

3.4.3 Plan Hierarchy Formation

As mentioned, the plan hierarchies are structures of behavior processes. Only one plan hierarchy is active at a time, and is the result of the highest priority score on a plan-generating process. The leaf nodes of the plan hierarchy are action processes, allowing the system to take action in pursuit of the goal at the root of the hierarchy. The typical non-leaf nodes of a plan hierarchy are condition processes and plan fragment processes.

Condition Processes

A condition process ("condition") is a bound behavior process that writes a boolean to its schema's interaction history reporting whether a specified attribute of its schema(s) is true or not ("yields true" or "yields false"). The condition process is responsible for continuously checking on the state of its specified attribute, and propagating priority scores appropriately. Each condition is bound to one or more schemas, usually of different types. Examples of conditions and their schema bindings:

- Whether the hand is near a specified object (comparing an attribute value between an body schema and an object schema).
- Whether the hand is far enough above the table to move without hitting objects (comparing an attribute value betwen an body schema and a location schema).
- Whether an object is in a specific location (comparing an attribute value between an object schema and a location schema).
- Whether the hand is holding a specified object (body schema and object schema).
- Whether the head is looking at the table (body schema and a hand-coded property).

A condition process can receive a priority score, just like any other behavior process. As mentioned above, a primary motivation can instantiate a condition process, in an effort to make the condition become true.

When a condition process has a positive priority score and yields true, then the primary motivation that instantiated it should be satisfied and the condition and any attached hierarchy can be deleted. Otherwise, if a condition has a positive priority score and yields false, it is the role of the planner to produce a plan hierarchy for addressing the condition. The planner is a limited resource, so conditions compete to be addressed by the planner based on their priority scores, just as other behavior processes compete for resources like arm and hand control in order to execute.

Each time the main loop of the planner executes, it finds the unsatisfied condition with the highest priority score and attempts to attach a suitable plan fragment process as a child node to the condition. Plan fragment processes (described in the next section) include a list of effect conditions that are expected to become true if the plan fragment executes successfully. Thus, the construction of a hierarchy begins with the back-chaining of a plan fragment to the condition node, based on the expected effects of the plan fragment.

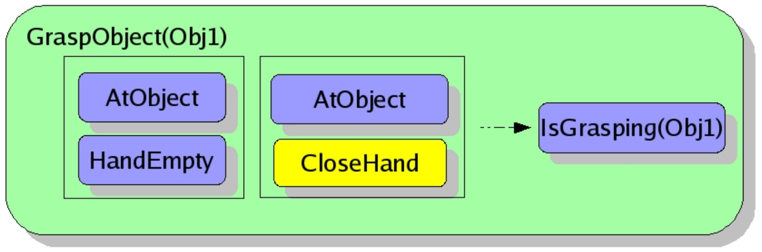

Figure 3-6: Illustration of the internals of a plan fragment process. The plan fragment process includes a sequence of condition and action processes, and also encodes an effect. In this example, the plan fragment for grasping an object includes two conditions in its first sequence step, which require the hand to be at the object and to be empty. When both conditions in the first sequence step yield true, the plan fragment can continue to its second step, which still requires the hand to be at the object, but now activates an action process to close the hand. The encoded effect, that condition `IsGrasping(Obj1)` will be satisfied, is used for backchaining in the planner.

Once the child node is attached, the condition then passes its priority score to the child until the child either fails or completes its execution. When passing its priority score to its child, the condition retains none of its priority score, thus removing it from further consideration by the planner.

Plan Fragment Processes

A plan fragment process ("plan fragment" for short, not to be confused with "plan hierarchy") is a bound behavior process that outwardly resembles an action process in that, when activated, it attempts to execute some behavior, and then monitors the failure or completion of that behavior. The difference is that action processes handle simple commands sent to the motor control modules, while plan fragments coordinate one or more action processes and conditions.

Each plan fragment consists of a sequence of steps, and within each step is a set of children, representing sequence elements. The children can be conditions, actions, or other plan fragments, all bound to the same schemas as the parent plan fragment. Each plan fragment subclass includes a specification of how to bind its schemas to its children. A example of a plan fragment process is depicted in Figure 3-6.

When the plan fragment receives a priority score and is active, it propagates its priority score to each child in the first step of its sequence. Each of those children receives the priority score and uses it to compete for its own activation; the plan fragment retains none of its priority score. When the child is a condition, it competes to be examined by the planner. When the child is an action, it competes to execute. Finally, when the child is another plan fragment, the nested plan fragment activates its first set of elements as well.

As soon as all children in a step are *satisfied* (conditions report being true, actions and plan fragments report being done), the plan fragment deactivates the children in that step and activates those in the subsequent step.

This sequential activation allows the plan fragment to coordinate a series of actions and conditions over time. By including conditions, the plan fragment can require specific conditions to be true before a specified action is taken. Furthermore, the inclusion of conditions enables planning to chain preconditions to the plan hierarchy in order for actions within each plan fragment to execute correctly. The plan fragment also encodes the effect conditions that are expected to become true if it executes successfully.

Thus, the plan fragment is essentially the operator of a STRIPS-like planning system, representing an action along with its preconditions and effects. Because the plan fragment allows any number of conditions and actions to be specified in an arbitrary sequence, it is more flexible than conventional STRIPS-style schemas.

An example of this is using a plan fragment to specify not only a condition that must hold true before a specified action (precondition), but to specify that the condition must hold true for the entire duration the action is executing (sustained condition), and then that the condition must be true afterwards as well (postcondition).

The arbitrary length of plan fragments also allows actions and conditions to be chained together, always executing as a continuous chain. This can save computation time in the planning algorithm for chains of actions that always occur in sequence, by sequencing actions that are always performed in sequence, rather than planning each step individually.

Plan Execution

The result of attaching plan fragments as children to conditions, and then instantiating the children of the plan fragments, is a plan hierarchy. Because priority scores are passed from parents to children in the hierarchy without being retained by the parents, only the current leaf nodes have positive priority scores at any given moment. When the active leaf node is a condition, it uses its priority score to compel the planner to make suggestions for it. When the active leaf node is a plan fragment, it activates its first set of children. When the active leaf node is an action, it uses its priority score to compete for resources to execute its programmed behavior.

Condition Backchaining

Backchaining is a term that describes planning to achieve an effect by finding an operation that lists the desired effect among its postconditions, and then adding that operation before the desired effect in an plan sequence. Plan hierarchies are the result of backchaining on conditions with high priority scores.

The method for backchaining a condition is as follows:

1. The planner generates the list of all potential plan fragment processes that could be bound to the same schemas as the targeted condition.

2. By examining all listed effects for those potential plan fragments, all plan fragments capable of addressing the targeted condition are added to a suggestions list for that condition.

3. Each plan fragment comes with an expected *success rate* (see next section) for completing the targeted condition. The condition makes use of these success rates to select a plan fragment from the suggestions list.

4. The condition selects the plan fragment with the highest priority score when adjusted by the expected success rate. This plan fragment is instantiated and activated, and becomes the child node of the condition.

5. Once the condition has a child node, it gives up its priority score to its child and falls off the list for planner consideration.

Because plan fragments encode preconditions and effects for the purpose of planning, all simple action types are wrapped in a plan fragment in order for the planner to make use of them. This means that, for every simple action (such as reaching towards an object, or closing the gripper), there exists a plan fragment that contains only that action, the preconditions of the action, and the set of effects of the action.

3.4.4 Predicting Success Rates

Each plan fragment subclass is associated with an interaction history that includes a record of the successes and failures of each instance of that subclass. Examining successes and failures across this history allows the system to predict the likelihood of success outcomes of future instances, given the attributes of a target object schema. This information is used by the planning system to decide on courses of action, and amounts to computing the expected affordances for a given object based on its attributes.

After each loop cycle of a plan fragment process, the process can declare a success outcome, a failure outcome, or a continuation. A plan fragment declares success if its sequence elements have all completed successfully. A plan fragment fails if any of its sequence elements declare failure. Most of the time, the plan fragment is still waiting for a sequence element to complete, so it asks for a continuation, and it will execute its loop again during the next cycle.

When a plan fragment declares either success or failure, the following set of data is stored by the interaction history database:

- `plan_fragment_subclass`, a string label describing the subclass of the plan fragment.
- `slot_name`, a string label describing the binding slot occupied by a given schema. For instance, in `Grasp(object = Obj1)`, `Obj1` is a schema bound to a plan fragment of type `Grasp`, via binding slot `object`.
- `attribute_field`, one of the attribute fields of the schema, such as color or shape.

- `attribute_value`, the value of the attribute given by `attribute_field`. If the object represented by schema `Obj1` is green, then the database would store `color` for `attribute_field` and `green` for `attribute_value`.
- `elapsed_time`, the amount of time in seconds since the plan fragment was activated.
- `outcome_boolean`, `true` if the plan fragment was successful, `false` if it failed.

Every applicable tuple is saved to the interaction history database for each success or failure of a plan fragment. This means that an entry is recorded for every attribute of each schema bound to the plan fragment, one for color, one for shape, one for location, and so on.

When it comes time to predict the likelihood of success of a new plan fragment process, the previous successes of plan fragments of the same subclass bound to object schemas with the same attributes can be looked up and used to assess the likelihood.

For instance, if the plan fragment

```
MoveObjectTo(object = Obj1, location = Loc2)
```

is suggested, the object schema `Obj1` has categorical attributes `color = red` and `shape = square`, and the location schema `Loc2` has the categorical attribute `out_of_reach = true`, then the set of interaction history lookups for this suggestion will be:

```
- (MoveObjectTo, object, color, red)
- (MoveObjectTo, object, shape, square)
- (MoveObjectTo, location, out_of_reach, true)
- (MoveObjectTo, object, id, Obj1)
- (MoveObjectTo, location, id, Loc2)
```

The last two entries permit the system to record interactions with a specific object, so regardless of any expectations derived from the attributes of an object, outcomes with one particular object can be predicted.

In addition, each plan fragment subclass also accumulates a baseline measurement of success and failure for lookup:

- (MoveObjectTo, null, null, null)

The baseline measurement records how often the plan fragment subclass succeeds regardless of its schema bindings, and is useful when insufficient data has been taken for adequate predictivity based on specific attribute values.

This set of lookups will produce the list of prior interaction results with the given set of attributes. For each set of interaction history lookups, the list of elapsed times and successes are collected. Then, for each lookup, it is possible to compute a mean and a variance on the *outcome time*, which is the amount of time previous attempts took to succeed, as well as the *success likelihood*, which is the proportion of outcomes that end in success, given that particular lookup.

In the given example, suppose that the interaction history for `MoveObjectTo` indicates:

1. a large variance of outcome times and a 40% success likelihood for `color` being `red`,

2. a 2% success likelihood and a large variance of outcome times for `out_of_reach`,

3. consistently short success times around 30 seconds, and an 80% success likelihood for `Obj1`, and

4. the system has never encountered an attempt to perform a `MoveObjectTo` on anything `square`.

It can be concluded from this that:

1. the outcome time based on `red` provides relatively little information because of the high variance,

2. the success likelihood based on `out_of_reach` provides good information because it is so low,

3. there is similarly good information predicting a high rate of rapid success for `Obj1`, and

4. no information comes from the object's being `square`.

The appropriate way to formulate a result prediction for this instance of `MoveObjectTo` is by using the past success likelihoods and the variance in outcome times associated with each lookup tuple. The final success likelihood, p_s is the minimum of all success likelihoods given by the lookup tuples, with some amount of back-off for lookup tuples with low occurrence. The final outcome time, t_s, can be computed as a weighted average of the mean outcome times given by the lookup tuples, where the weights are determined by the "sharpness" of the variance and the number of occurrences of that lookup tuple. As a simplification, the computation of outcome times in the current implementation pessimistically takes the maximum outcome time given by any lookup tuple, provided the variance in outcome time and the occurrence count are sufficiently low and high, respectively, relative to hand-coded thresholds.

In the current example, the information based on `red` is ignored because it has a high variance. The success likelihood based on `out_of_reach` is the lowest success likelihood in this set of lookups, so it serves as the success likelihood for this prediction. However, the timing information from `out_of_reach` is useless because of high variance and infrequent occurrence. The rapid, low-variance success of `Obj1` gives the only useful timing information, around 30 seconds.

The prediction mechanism needs to incorporate both the success likelihood and the outcome time into a single value called the *success rate*, which describes the expected amount of time, in seconds, before the plan fragment completes successfully. In order to produce a final success rate, processes are modeled as Poisson processes, which always have the same exponential probability of success at any given moment. This is like a light bulb, which could blow with the same probability at any moment, and thus it always has the same expected time to failure.

For the given plan fragment, the interaction history has provided a likely amount of time to success (30 seconds). Then, the success likelihood (2%) is taken into account by assuming that at the 30-second time that the process is likely to complete, it will either succeed or fail with that 2% probability of success.

Then, the success rate is the time to success divided by the success likelihood: $t_r = t_s/p_s$. Thus, for a process that is quick to succeed when it succeeds, but with a poor success likelihood, the actual expected time to success will be proportionately longer. In this example, $t_r = 30/0.02 = 1500$ seconds.

The amount of time derived from this calculation is then sent through a hyperbolic discounting equation: $k = 1/(1 + t_r/\tau)$, where t_r is the computed success rate, and τ is a hard-coded time constant representing the system's "patience." k then becomes the factor that the priority score is multiplied by to adjust it for its computed success rate.

If a behavior has a priority score of 1 given immediate completion, then it will have an adjusted priority score of 0.5 if it completes in τ seconds. This causes conditions to consider suggestions based on both their priority score, as adjusted by the primary motivations, and also the predicted amount of time until the suggestion completes.

For $\tau = 300$ (five minutes before priority score drops by half), in the given example, $k = 0.167$. This means that even if the primary motivation supporting the `MoveObjectTo` supported the behavior with a maximum priority score of 1.0, the adjusted priority ends up only being 0.167 due to the expected success rate. This reflects the unwillingness of real animals to engage in extremely low-probability behaviors.

3.4.5 Recap

The computation of success rates gives the planner the ability to deal with cases where the condition process being targeted for hierarchy expansion can be addressed by more than one plan fragment. By computing success rates based on the interaction history of objects with similar attributes to the object targeted by the condition process, the planner can make use of predicted affordances of otherwise unfamiliar objects in making its decisions.

Because attribute values such as `red` or `square` are used as indices for success rate prediction, one significant aspect of the model is that categorization of features is not merely useful for linguistic labeling (connecting words to attributes) and object identity tracking (using the color of an object to follow it). The categorized attributes also drive the ability of the system to predict results of interactions with unfamiliar objects.

To review the mechanisms behind the plan hierarchies: the primary motivations suggest and set priority scores of condition processes and plan fragment processes. Plan fragment processes are sequences of other processes, which they instantiate and pass priority scores to sequentially. Condition processes compete based on priority score to be addressed by the planner. The planner examines each condition, finds plan fragments with appropriate effects for the condition, and predicts the success rates for each plan fragment. Based on the success rates, the planner selects one plan fragment to attach as the child of the condition. The hierarchy is thus composed of alternating conditions and plan fragments, down to the leaf level, where the plan fragments create action processes as their children, which take action rather than linking to further children.

3.4.6 Revising a Plan Hierarchy

A plan hierarchy is composed of any number of conditions, actions, and plan fragments. Each action and plan fragment in the hierarchy can succeed or fail at any time, and each condition can yield true at any moment. As these successes and failures occur, the interaction histories for the plan fragment subclasses are updated. Furthermore, the state of the environment can change outside of the robot's direct control, either due to noise, error, or human intervention.

The decision-making in each plan hierarchy occurs at the condition nodes. Each condition node supports its child only until the condition yields true, or until the interaction history for its child suggests that the child will not succeed with the success rate initially predicted.

If a condition becomes true in the middle of execution, it deactivates its child and passes its priority score back up to its parent. In the case that its parent is a plan fragment, this may enable the plan fragment to move on to the next sequence step. In the case that the condition is the root of its plan hierarchy, the responsible primary motivation is notified and the plan hierarchy is deleted.

If a condition has a plan fragment child and the plan fragment fails, the condition discards the plan fragment and re-enters the competition to be addressed by the planner. The planner will reassess appropriate plan fragments and will attach one to the condition. Typically, this leads to the same plan fragment subclass being reinstantiated, reattached, and run repeatedly, but the updating of the interaction history can cause the success rates to vary until a different plan fragment subclass is suggested instead.

If a plan fragment runs for a long time and its interaction history suggests that the initial success rate prediction is overly optimistic, the condition will also reclaim its priority score and compete to be addressed by the planner again. When this happens, the planner will once again check the effects of available plan fragments for suitable plan fragments and make suggestions. If a suggested plan fragment is substantially better in expected success rate than the current child, the condition will deactivate the child, even if it was continuing to run without declaring success or failure, and activate the new suggestion.

These revision steps enable the system to respond to real-time events and results as they happen, based on how they affect the currently active plan hierarchy. It allows the system to maintain a certain level of reactivity even when using plans to pursue goals.

3.4.7 Sensory Processes and Reaction Control Processes

Two other types of processes sometimes occur in plan hierarchies by virtue of being in a plan fragment's sequence. Sensory processes are behavior processes that interpret and record sensory information to interaction histories, and reaction control processes enable some control over unbound reactive processes in the context of a plan.

Sensory processes usually run persistently as part of an object schema. Examples include the tracking processes that write color and location information to a schema's interaction history. Sometimes, however, the sensory process is part of a plan fragment sequence, in which case the sensory process is limited to running when that plan fragment reaches that point in its execution. The key example of this is weight measurement, which is only meaningful when an object has been grasped and lifted. Thus, as part of the plan fragment that lifts and moves an object, the weight of the object is measured by reading the forces at the robot's fingertips.

Reaction control processes enable the current plan hierarchy to serve as a context for controlling unbound reactive processes. For instance, collision-detection processes are always running, and the collision motivation will always use detected collisions as a trigger to move the robot away from a collision. However, if the robot is attempting to grasp an object, some level of fingertip force beyond the usual collision threshold would be expected, in order to get a firm grasp. A reaction control process is like an action process, except that its execution leads to the alteration of the collision thresholds in the reactive processes, rather than a direct action. When priority score is propagated to a reaction control process, it makes the changes to the appropriate collision thresholds until it loses its priority score, which occurs when its sequence step is completed.

The presence of both sensory processes and reaction control processes within a plan hierarchy are other ways in which the planning system directly affects the other perspectives. In the case of sensory processes, the plan directs the writing of data to the interaction histories, and in the case of reaction control processes, the plan alters the reactive behavior of unbound, always-on processes.

3.5 Perspective 4: Language

Perspective 4 involves processes that are created or activated according to language input, in order to handle requests from the human. Language input comes in two basic forms: descriptive, and directive. Descriptive speech acts inform the system of some attribute of an object, such as “The red ball is heavy.” Directive speech acts inform the system that the human desires an action to be taken, such as “Pick up the red ball.”

The model receives verbal utterances in the form of a parse tree of tokens that translate into various structures within the model. Verbs in the utterance are represented by verb tokens in the parse tree, which then lead to plan fragment processes. Noun phrases in the utterance are represented by noun phrase tokens in the parse tree, which then lead to reference processes. Description terms (such as colors and shapes) in the utterance become description tokens attached to noun phrase tokens in the parse tree, and are stored in the reference process.

3.5.1 Reference Processes

Both types of speech acts (descriptive and directive) make use of *reference processes*, such as the process that handles “the green block.” A reference process is a behavior process that can also act as a stand-in for binding other behavior processes to an object schema. Condition, plan fragment, and action processes can all be bound to the reference process as if it were an object schema, and as soon as the reference process *resolves* its *referent schema* (or just *referent*), all its bound processes will act as if they were bound to the referent schema.

This resolution is performed by searching for object schemas with categorical attributes that match the description tokens of the reference process. The description tokens are constituents of the noun phrase token in the parse tree. They derive from adjectives and nouns in the utterance, such as color words, shape words, and weights like "light" or "heavy."

In the example of "the green block," "green" and "block" become description tokens that are stored by the reference process. As depicted in Figure 1-3 back in Chapter 1, the reference process resolves itself to the object schema with matching categorical attributes. The reference process then dynamically rebinds all its bound processes to this referent schema, and behaviors then execute as usual. If no referent schema is found within a certain time, then all its bound processes automatically fail. The rebinding of a behavior process to a referent schema is depicted in Figure 3-7.

The reference process needs to compete with other behavior processes to use the resources necessary for its resolution. For instance, resolving a reference process might require looking at the table to identify the current set of objects, so if the head is facing the human, the reference process acts like a condition process, competing to create a plan hierarchy to look at the table.

The reference process also becomes a precondition for any plan fragment that is bound to it, which means that the plan fragment cannot proceed, and none of its actions can be executed, until the reference process is resolved. After all, it would be meaningless to execute an action or plan fragment towards an object that has not been identified.

3.5.2 Plan Fragments

Directive speech acts use action verbs, such as "pick up" and "hand me." These verbs are passed to the model as tokens that translate into plan fragment processes representing the associated actions. Each plan fragment is then instantiated and given a priority score by the social motivation, leading to plan hierarchy construction. The plan fragments that represent the actions have appropriate preconditions (sequence slots) built-in. For instance,

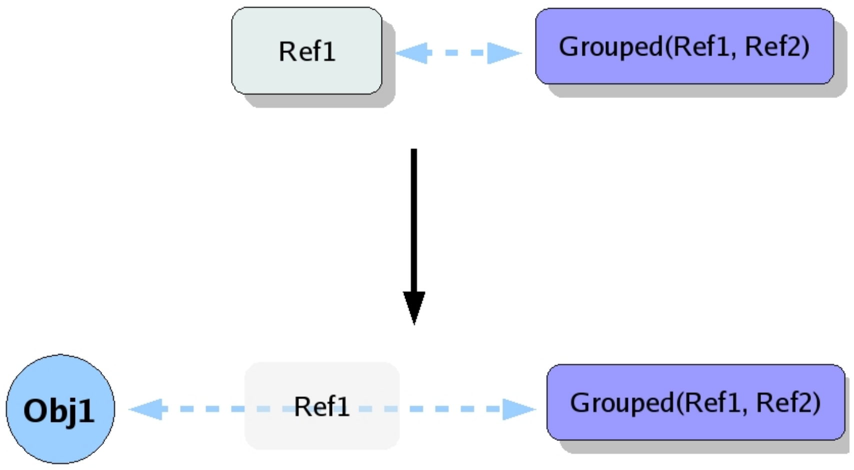

Figure 3-7: Depiction of a reference process rebinding a bound behavior process. At top, the condition process "Grouped(Ref1, Ref2)" is bound to reference process "Ref1" as if Ref1 were an object schema. Ref1 is searching for a referent schema, and the condition process cannot proceed (in this case, be used for planning) until Ref1 has resolved. At bottom, Ref1 has found its referent schema, and the condition process is rebound via the reference process to object schema Obj1, and subsequently acts as though it were bound to Obj1.

the plan fragment that handles "pick up" has a precondition that the robot be successfully grasping the object.

Descriptive speech acts use linking verbs such as "is" and "are." In this case, a plan fragment is still necessary in order to provide a precondition to ensure that the reference process is resolved. For instance, "the red ball is heavy" can only be handled after the reference process for "the red ball" is resolved. The sequence of the plan fragment is simpler than for action verbs, though; it instantiates a sensory process whose role is to write the categorized value, in this case `heavy` to the interaction history for the referent schema.

In both directive and descriptive speech acts, the plan fragment is bound to the reference process or processes that result from the noun phrase tokens in the incoming parse tree. The resolution of the reference process becomes the first precondition for any such plan fragment process.

Thus, speech interaction is handled by the language system using a variety of interactions with the belief context and the planning system. Reference processes act as stand-ins for object schemas, and can be bound to behavior processes. The reference process takes action by searching for its referent schema based on its description tokens, and acts like a condition process in plan hierarchies to ensure its resolution. Finally, actions are handled as plan fragments, supported by the social motivation, and otherwise dealt with in the same way as normal plans.

Speech output is produced by a set of action processes, which are incorporated into plan fragments appropriately. Typical speech outputs for the model are simple acknowledgments ("Okay"), failure notifications ("I can't do that"), and object descriptions, which are handled by a simplistic hand-coded system that makes use of the category labels within the interaction history. Object schemas to be described by the action processes that produce verbal descriptions are bound to the verbal description action processes the same way all schemas are bound to behavior processes.

3.6 Summary

This chapter described the four perspectives of the OSM, explaining how schemas, behavior processes, primary motivations, the planning system, and verbal interactions work together. Object, location, and body schemas consist of behavior processes that act, sense, and react to both the real world and to the history data stored within the model. Meanwhile, the schemas themselves serve as stabilized constructs for planning and language use. The planner generates plan hierarchies to pass priority scores between behavior processes, and makes use of interaction histories to predict object affordances in order to select plan fragments. Verbal processing leads to new behavior processes that interact with the belief context and the planning system to generate plans and resolve referents for carrying out verbal instructions.

Chapter 4

Implementation on an Embodied Robotic Manipulation Platform

Building an instance of the model on a robot that acts in the real world requires integration of a diverse array of hardware and software modules. This chapter explains the components of the system, along with the capabilities they enable, in greater depth. Then, I will narrate in more detail one of the example interactions from Chapter 1.

4.1 A Physical Robotic Manipulation Platform

The robot used for this implementation, named Trisk, has an arm with six degrees of freedom, capable of moving its end effector in the space above a table placed in front of the robot. The end effector of the arm is a Barrett Hand with three fingers. Each finger has one degree of freedom, and the whole hand has a spreading mechanism that enables two of the fingers to either oppose the remaining finger, or to move in parallel with the third finger. The three fingers and the spreading mechanism together constitute four degrees of freedom. Figure 4-1 shows the robot, and the hand in the two positions offered by the spreading mechanism.

Separated from the arm, a head assembly (see Figure 4-2) is mounted on a robotic neck with four degrees of freedom, enabling the head to face various directions at various angles, including downwards towards the table and forwards towards the rest of the room. In the

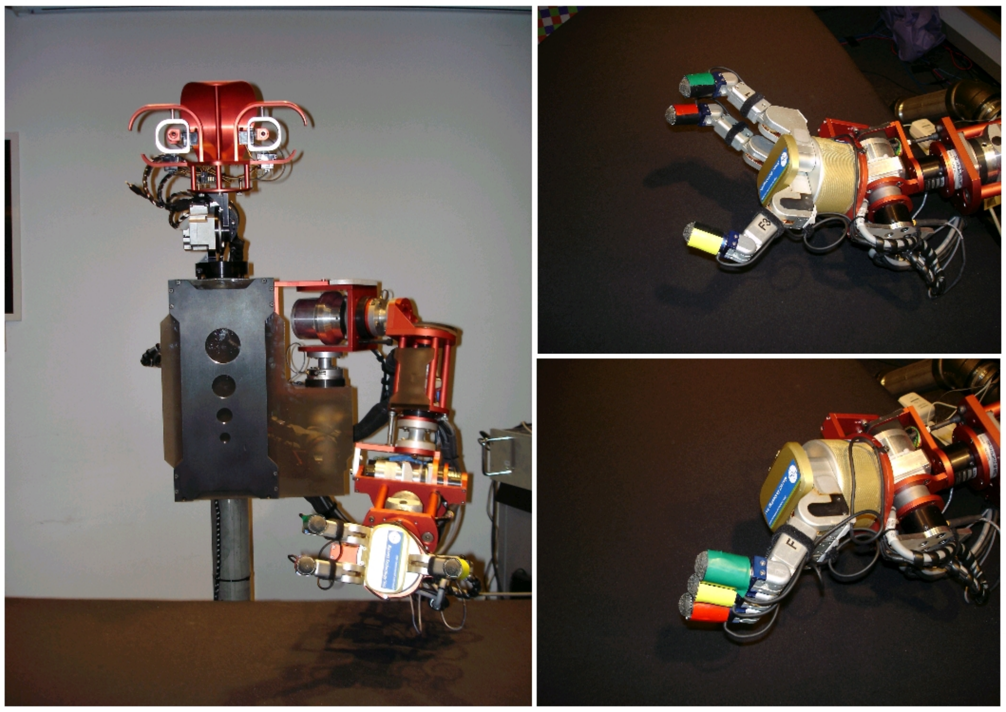

Figure 4-1: Left: Trisk is a robotic manipulation platform with a 4-DOF neck and 6-DOF arm mounted on a common torso. Right: The 4-DOF Barrett Hand mounted on the end of the arm can move each finger with one degree of freedom, and change the spread between its fingers with the remaining degree. The top right image shows the fingers opposed and prepared to grasp, while the bottom right image shows the fingers close together, which is good for tapping objects.

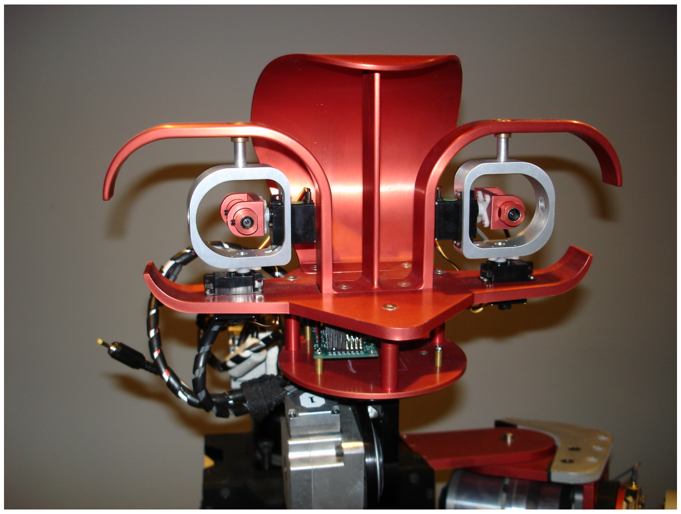

Figure 4-2: Trisk's head, mounted on the moving neck, has two cameras each mounted in a 2-DOF servo assembly.

head are two cameras, each mounted in a two-axis servo assembly, enabling the cameras to rapidly reorient in approximately a 160-degree range in two perpendicular axes relative to their mounting points on the head.

The arm and the neck are both mounted on a torso assembly that is anchored to the floor. The robot is positioned facing the table such that the hand can be moved to touch, grasp, and lift objects within view of the cameras when the head is facing downwards.

The sensors in the robot platform provide the feedback for behavior processes to guide actions. The two cameras provide video input for visual processing. The hand provides position information for its four degrees of freedom, and each fingertip has a sensitive six-axis force-torque sensor that can be used to determine contact position and force of any surface contacting each finger.

On the arm, each of the six joints has an encoder that reports position information and a load cell that reports the amount of torque on the joint. The torque provided by the load cell enables torque-based control of the arm, which makes it compliant to external forces while active, and also enables it to report the amount of torque being used to move to or sustain a target position. Neither the hand nor the neck use torque control, so they are limited to position-based control only.

4.2 Computational Resources

The software side of the system is run on a network of approximately 15 off-the-shelf computers with CPU's between 1.0 and 3.0 gigahertz. The computers run Linux, with the exception of an OS X Macintosh doing speech production and a Windows machine for interfacing to the neck hardware control board. Several computers are dedicated to hardware-specific tasks; each camera has a dedicated computer for receiving video input, the camera mounts have a computer that sends commands to the servo control board, audio input and output are handled by specific machines, and the arm and hand each have a dedicated computer for real-time control and state.

4.3 NetP and Network Architecture

Other processing and integration tasks are spread across ten computers, all connected via the PVM software architecture for coordinating heterogenous distributed network clusters. Our group's processes use a networking layer written on top of the low-level PVM code, called NetP [39]. NetP enables communication between processes spread across multiple PVM-networked machines using a channel abstraction, in which a process broadcasts messages to a channel, and multiple processes can subscribe to that channel and receive the messages. Messages are structured within mapping and list containers.

NetP provides an API that can be called from C++, Python, and Java, running in Linux, OS X, and Windows, making it appropriate for coordinating relatively complex heterogeneous distributed systems in which every module has different computational needs and characteristics. As such, the code for the various modules is written in a mix of C++, Python, and Java, with a tendency towards using C++ for low-level real-time control modules, Python for high-level integration, and Java for graphical output and simple control tasks.

4.4 Hardware Interface Modules

In this section, I describe software modules that communicate directly with the hardware for performing motor control and acquiring state and sensor information.

4.4.1 Arm Control and State

The direct control of the robot arm is performed by a real-time process (the *control process*) with access to the analog-to-digital and digital-to-analog converters that connect to the robot's motor hardware controllers, encoders, and load cells. The control process runs a two-layer PID control loop on joint position and torque, sending motor control signals to produce appropriate torque at each joint to arrive at a targeted set of joint positions. Joint positions are specified by a second process (the *host process*), connected over the network, that provides sequential trajectory lists of joint positions for the control process to target. A trajectory list is a list of relatively adjacent joint positions that moves the robot smoothly from one position to another over a period of several seconds.

The arm's host process uses NetP so other modules can request motions for the arm over the network. The host process handles inverse kinematics, so other modules can specify rotation matrices in 3-dimensional Cartesian space as targets for the end effector, and the host converts these targets to sets of joint angles, produces a smooth trajectory list, and sends it to the control process to be executed. The host process also takes the current state information for the robot, including joint positions and torque, and broadcasts it over NetP for other modules to process. The host process also handles forward kinematics, by converting joint positions of the robot to Cartesian coordinates.

4.4.2 Hand Control and State

The control of the hand is handled similarly to the control of the arm. A control process handles real-time control and state information, and a host process uses NetP to receive requests for action and distribute the state information. Control of the hand is simpler than that of the arm, because the hand itself only permits position control. The host process for the hand is simpler as well, because forward kinematics for the hand, including processing of contact force and position for the fingertip sensors, depends on the arm's kinematics, and is thus handled by a separate integration process.

The actions performed by the hand on demand include opening the spread motor in preparation to grasp an object, closing the spread motor in preparation to tap an object, grasping by closing the fingers until force is detected at the fingertips, and opening the fingers to release a grasp.

4.4.3 Neck Control and State

The neck control system also approximately parallels that of the arm, but without torque sensing, so all control is done with position information. Like the arm, trajectories are generated and the neck's motor controller moves the neck through the trajectory over a set period of time, enabling smooth motion between target positions.

4.4.4 Camera Control and Image Handling

The camera mounts use a servo controller that rapidly moves the cameras to their target positions. The serial communications to the servo controller is handled by a process that accepts target positions over NetP and forwards the request over the serial line.

Visual inputs from the cameras are digitized using off-the-shelf TV capture cards connected to the S-Video outputs from the camera hardware. Although the use of two cameras has been successfully used by this system to provide stereoscopic object positions, the current implementation uses only one camera for simplicity.

4.5 Sensory Processing Modules

With the help of the hardware interface modules, additional processing is done on kinematic, force, vision, and audio inputs to provide meaningful sensory inputs to the model.

4.5.1 Arm and Hand Kinematics

Although arm kinematics is handled through the arm's host process, the integration of arm and hand kinematics is handled by a separate program. A complete visualization model of

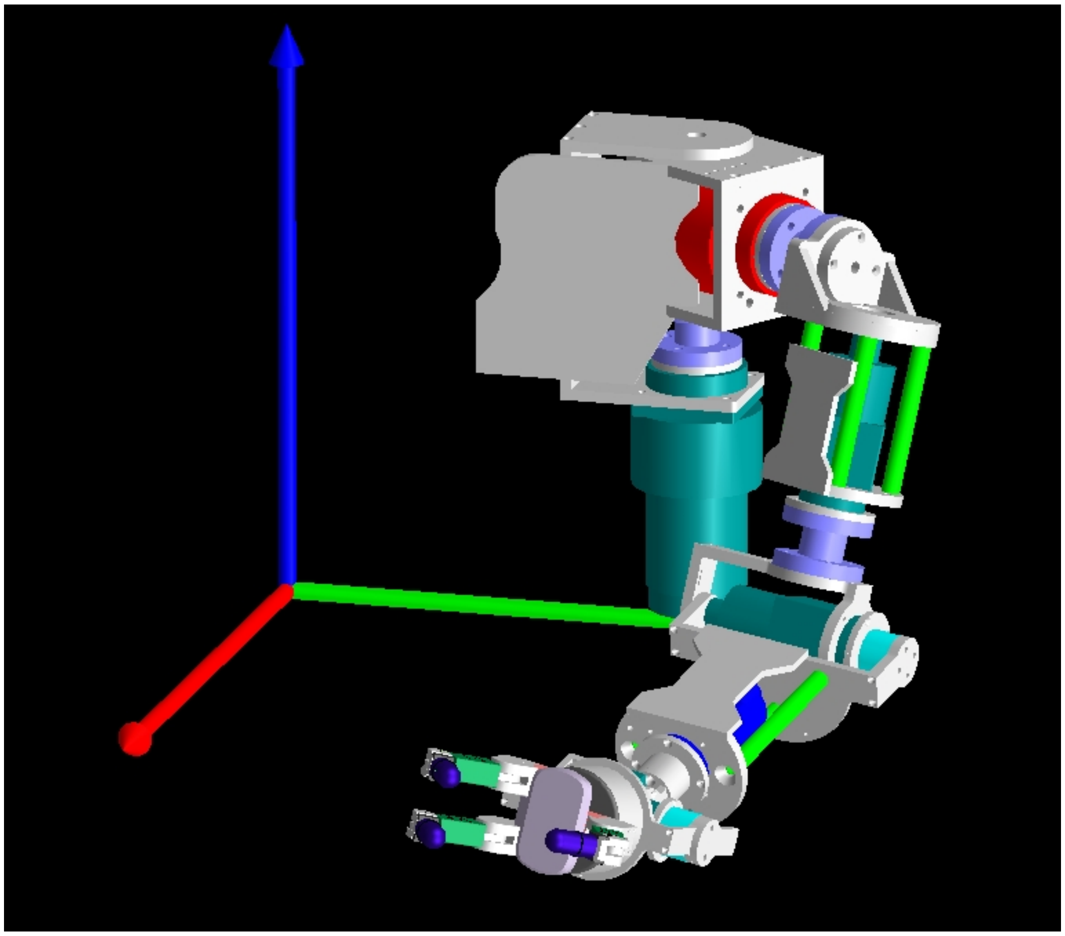

Figure 4-3: The 3-D graphical visualization of the Trisk's arm and hand state.

the robot arm and hand has been built in OpenGL, providing real-time 3-D visual state information about the robot's position and force interactions. This visualization process doubles as the integrated kinematics model; the connected hand and arm assembly within the model can incorporate joint positions from both the hand and arm host processes to produce Cartesian coordinates for each of the three fingers on the hand. Figure 4-3 shows the 3-D arm visualization model.

Furthermore, the visualization process takes the data coming from the six-axis force-torque sensors on the fingertips, and extrapolates the contact position and contact force on each fingertip. This information is then displayed on the visualization model, and also broadcast over NetP to provide contact force information for other modules in the system. Figure 4-4 shows examples of contact force information provided by the visualization.

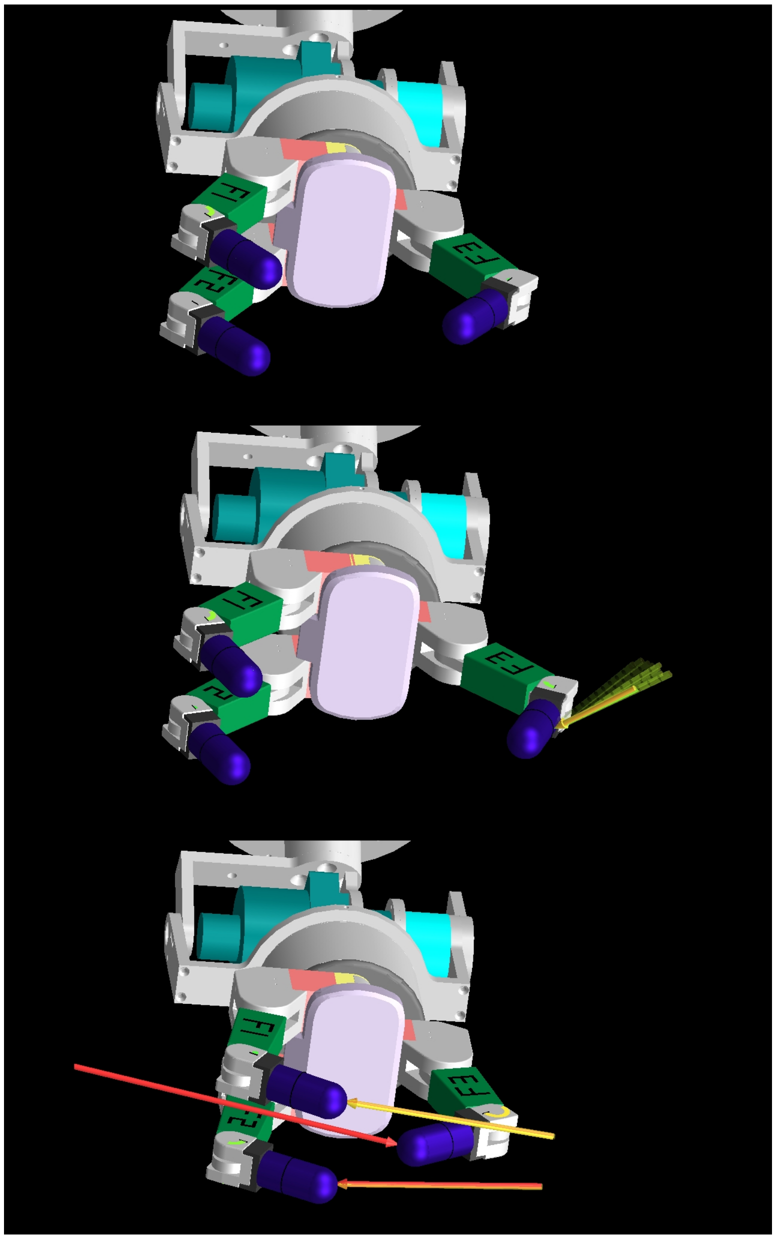

Figure 4-4: Close-up view of the hand in the 3-D visualization model. Top: The hand by itself. Middle: The hand, with a contact force pressing Finger 3 from the right. Bottom: The contact forces detected while grasping an object.

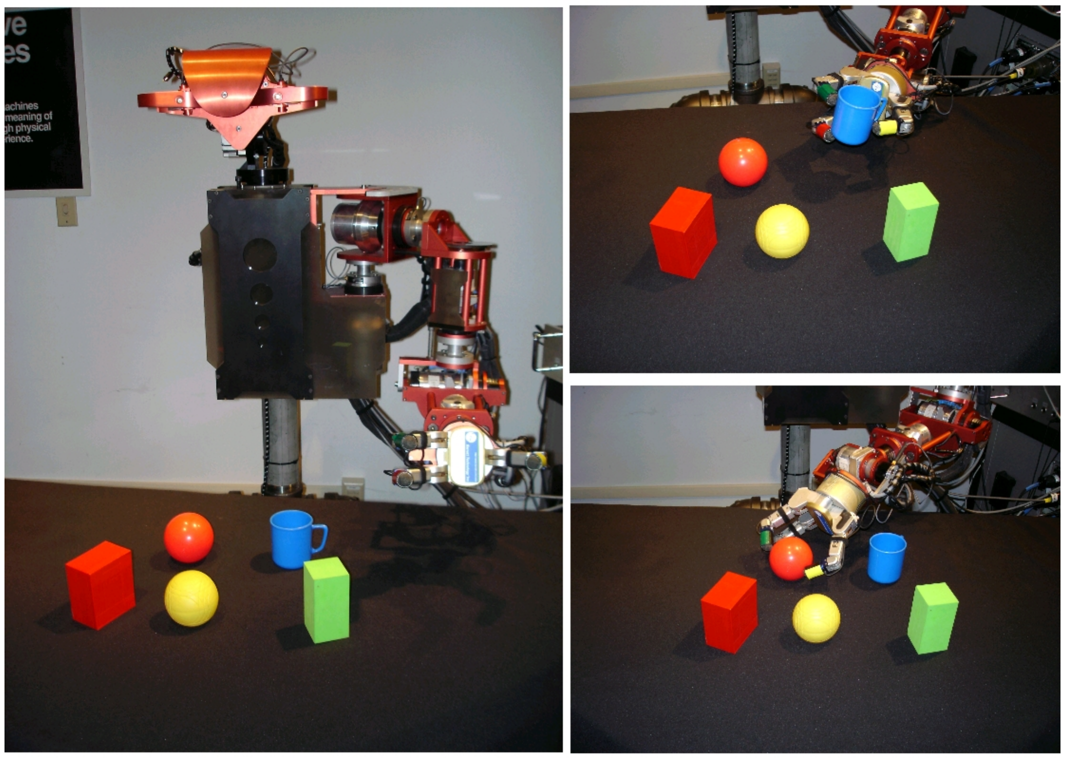

Figure 4-5: Left: Trisk looks down at a set of objects on the table. Upper right: Trisk performs a side-approach grasp of the blue cup. Approaching from the side is more reliable but more prone to collision with other objects on the table. Lower right: Trisk performs an above-approach grasp of the red ball. Approach from above is less reliable but avoids colliding the arm with the blue cup.

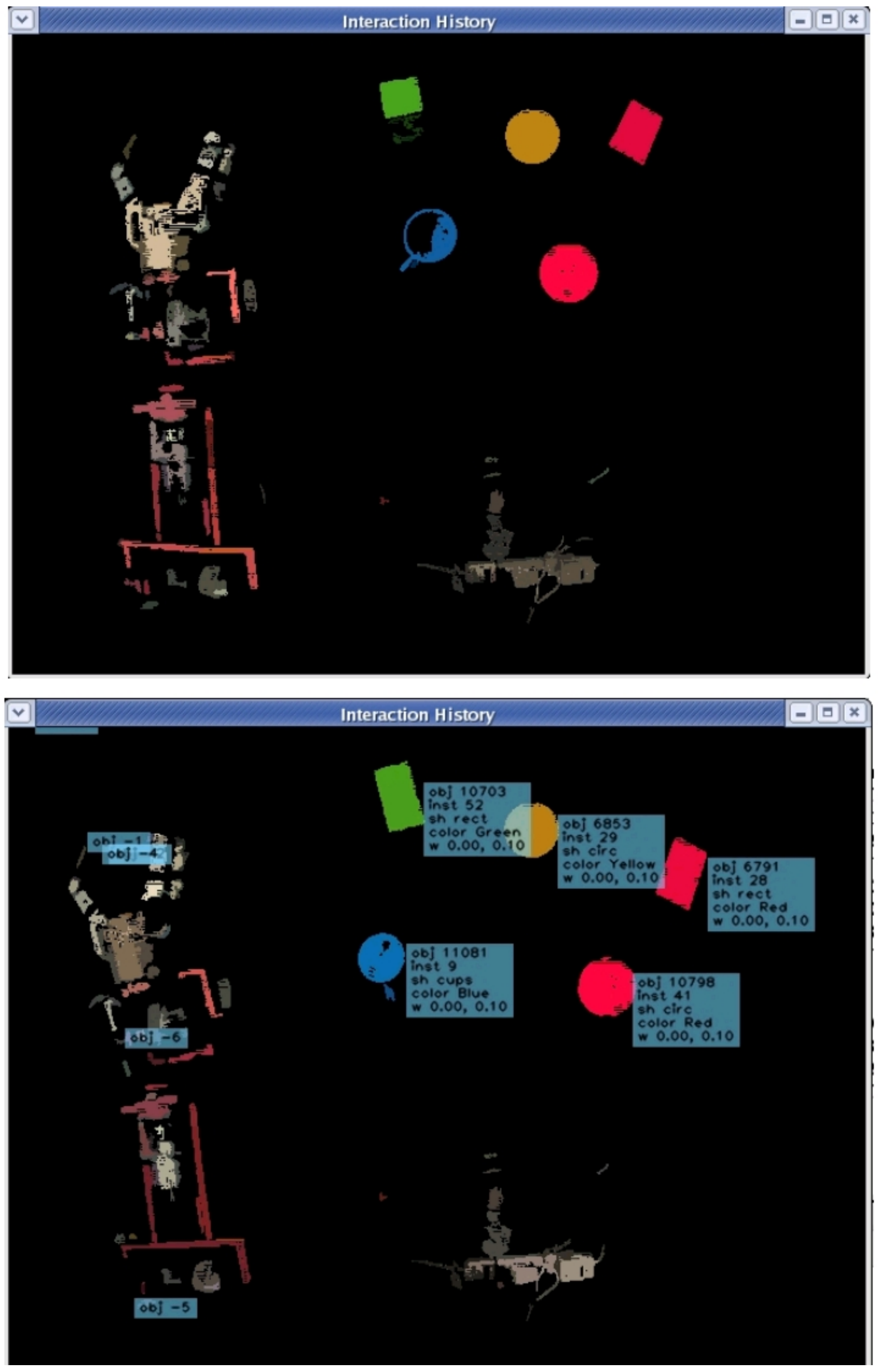

Figure 4-6: The visual segmenter processes the visual scene of the objects shown in Figure 4-5. Top: Initial segmentation of the image. Bottom: The segmenter display is also used for debugging output of the belief context. Each object schema is shown with some of its interaction history, and the locations of the arm parts and fingers are highlighted, after being processed by forward kinematics and the coordinate transformation processes. Knowing arm and hand locations is critical for predicting whether the arm might occlude a tracked object.

4.5.2 Visual Segmentation

Segmentation of visual input frames into likely visual objects is handled by a modified version of the CMVision software by James Bruce from Carnegie Mellon University [16]. The system performs a vector quantization of the YUV color space based on a set of labeled color category training examples. By producing a lookup table of continuous color values to quantized indices, each pixel in the input frame can be rapidly categorized as one of the known color categories. Then, the system groups adjacent pixels of the same category together in order to build the segments to be reported.

The training of the color models in the system is done by building Gaussian distributions on the set of labeled examples acquired by clicking on an input image and recording the colors and labels. The Gaussians are then used to classify every color in YUV space, which is then used by the CMVision system to build the lookup table and perform the segmentation.

Once the segmentation of the input frame is complete, the system converts the structure with the segments into a NetP list and broadcasts the centroid, size, and color of each segment.

The segmenter is also responsible for sending full segment information, including the pixel map of a segment, to other parts of the system, when commanded to over NetP. Furthermore, the segmenter is used as a simple graphical visualizer of objects in the system's current belief context. Segmentation and visualization of the belief context are depicted in Figure 4-6, based on the objects in Figure 4-5.

The output from the visual segmentation algorithm is used for the basic form of visual tracking in the system. The behavior processes assigned to do visual tracking can compare centroids, sizes, and colors in order to establish the revised visual attributes for an object schema in a new frame.

4.5.3 Visual Mean-Shift Tracking

The basic visual tracking of objects is performed using the output from the segmenter. However, the segmenter output is slightly fragile for tracking purposes due to factors such as noise, shadows, and occlusion; a sudden change in the reported properties of a segment can cause segment-based tracking to lose track of the object. More stable tracking is

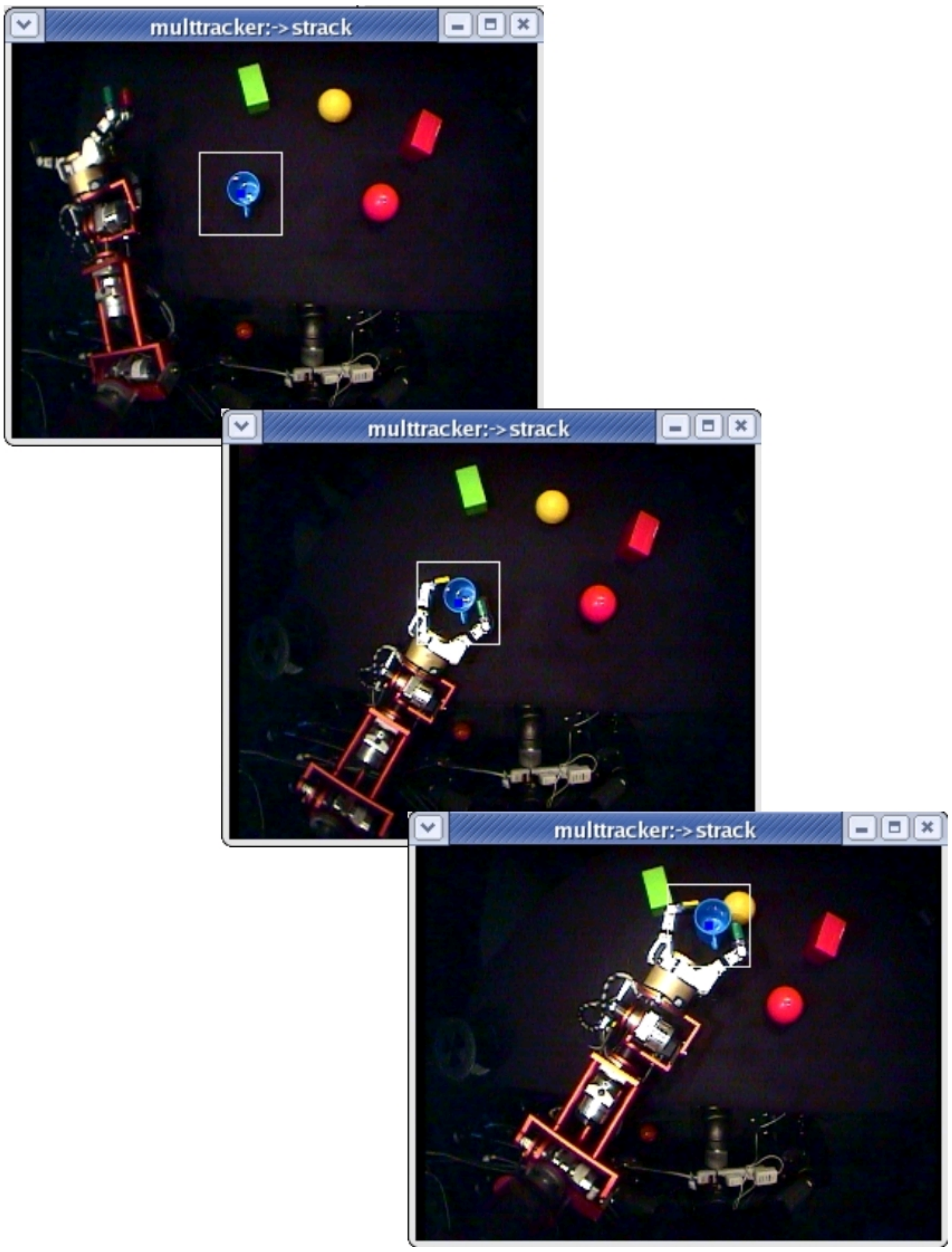

Figure 4-7: The mean-shift tracking module takes templates from the segmentation module and tracks them more robustly based on edge and color profile. Here, the blue cup is being tracked while being moved by the robot.

available to the system via an additional tracker that uses a mean-shift tracking algorithm [22] to follow one object in the visual field. This tracking is limited to one object due to computation time, and so it is typically reserved for a single object being targeted by a manipulation action. Running multiple mean-shift trackers across multiple machines in parallel would enable tracking of more, but still a limited number of, objects.

Mean-shift is an algorithm that iteratively searches a fixed-size window near the last known position of an object for the object's new position. Each point in the window is evaluated using a distance metric to determine the likelihood that the point is the new centroid of the object. The window is then moved so that it is centered around the weighted average position of all the likelihoods within the window, and the algorithm is iterated until convergence.

The distance algorithm used in our system is a weighted sum of edge gradient and color information. The segmenter is commanded to send a selected segment to the mean-shift tracker for tracking, and then the edge and color profile of this "template" segment is used to determine the likelihood of each pixel being the new centroid. The edge profile is determined by convolving a series of Sobel-like kernels (found in many textbooks, see e.g. [27]) across the template to determine the magnitude of its vertical, horizontal, and diagonal (for two diagonal directions) gradients. The edges of the object tend to be the strongest gradient points. The color profile is the color value of each pixel in the template.

In order to match the template against the target window, the incoming image is Gaussian-blurred in order to remove local sampling effects, and then the distance is computed between the template and each position in the target window, where the distance is a weighted sum of the differences between the values of the four edge profile channels and three color channels. Each distance measures the likelihood that the template is seen centered at that point in the window. Mean shift is then used to shift the window to the new object position. Confidence can also be reported by returning the distance value of the centroid point.

Compared to tracking of segment properties, mean-shift tracking is more robust to noise, shadows, and occlusion. However, it is more computationally expensive and it often falls into local minima and then persistently reports the wrong object. Careful threshold adjustment and handling of the mean-shift tracker output is necessary to ensure accuracy of object tracking. A sample result of the mean-shift tracker is shown in Figure 4-7.

4.5.4 Shape Recognition

Visual shapes are identified using an implementation of *shape contexts*, a method for shape recognition developed at U.C. Berkeley [8]. To compute a shape context, a region containing a shape is run through an edge detector that extracts a fixed number of randomly-sampled key edge points. Each edge point is then placed in the center of a bulls-eye-shaped structure with three concentric circles and eight slices. The sections of the bulls-eye are used as buckets for a histogram; the sampled edge points are counted into histogram buckets based on where they fall in the bulls-eye.

Training and recognition thus proceed in similar ways. For training, the histogram is computed and stored for a given shape. For recognition, the histogram is computed for the target shape and then the distance is taken between the vector representing the histogram for the target shape and the vector of each trained shape. Based on these distances, a vector of shape probabilities can be generated, and the most likely shape is used by the system for the shape category.

4.5.5 Speech and Language

Audio input is taken using a wireless headset microphone, and the audio is processed using the open-source Sphinx 4 [23] speech recognition software, with a lexicon and language model appropriate for this particular domain. The recognized words are passed to the language parser, which is an implementation of the Earley grammar parser used in our group in previous language-grounding projects [36].

The parser finds the most likely grammatical parse for the input utterance. The output of the language parser is a tree structure identifying the parts of speech of each token in the utterance, which is then converted into structures within the Object Schema Model.

Speech output from the system is provided by an off-the-shelf text-to-speech engine. When verbal output is requested by a behavior process, the categories describing the bound object schema are translated into words and sent to the text-to-speech engine.

4.6 Behaviors and Object Schemas

The current implementation uses a simplified resource system by assigning all motor and speech actions to the same resource, in order to avoid deadlock between hierarchical process structures with multiple resource requirements. An additional resource handles the mean-shift tracking module, which can only track one object, and one more resource handles the planner for addressing competing condition processes. Further decoupling of the resources is an issue for future work, although it could also be argued that not moving and talking at the same time is good for behavior coherence.

The code for behaviors is spread throughout various code modules, written in a mix of C++, Java, and Python. The coordination of the whole system takes place in a single code module written in Python, which handles resources, planning, schema management, and the shared memory for the interaction histories. Many of the behavior types are also implemented within the coordination module, with simple behaviors implemented as methods and more complex behaviors implemented as Python generators. Generators allow a method to be called and then paused, with saved state, to be continued later, while staying within the same execution thread.

Based on activity within the coordination module, behavior processes are instantiated and activated both locally and across the network. Behaviors implemented within the coordination module are simply called as local functions, while behaviors implemented within other modules distributed across the network can be instantiated, activated, and deactivated via NetP.

4.7 Sensor- and Motor-Related Behavior Processes

The various sensor inputs and motor outputs are handled by a number of behavior processes. Some of these are bound, and others are unbound. Some of these processes are always running, and others are instantiated only when triggered by specific events. The always-on processes are activated at startup and allowed to run continuously.

4.7.1 Arm- and Hand-Related Processes

Behavior processes that deal exclusively with the arm and hand generally fall into four categories: 1) Robot state, 2) Contact/collision processes, 3) Location tracking, and 4) Object-directed action.

Robot State Processes

The locations of various parts of the arm and hand need to be written to the body schema interaction histories continuously in order to provide updated information to other processes throughout the system. These always-on sensory processes are the coordination processes of their respective body schemas.

Contact and Collision Processes

Several always-on unbound reactive processes check for indications of contact and collisions in the arm and hand. There are three types of contact/collision:

Location-based Each joint and link in the arm and hand is checked against a set of hand-coded collision volumes to determine if that part is in danger of colliding with other parts of the robot, or pressing through the table, or hitting nearby equipment.

Torque-based Each arm joint reports the amount of torque being exerted at that joint. A set of hand-coded thresholds is used to determine whether the amount of torque at that joint exceeds the typical amount used to move the arm between positions. Excessive torque is interpreted as a collision.

Touch-based Each of the three fingertips has sensors that report contact position and force. Small force magnitudes on these sensors count as touch contacts, and excessive force magnitudes on these sensors count as collisions.

If a collision is detected, data about the collision is written to the interaction history of the collision process. Upon reading this data, the collision motivation instantiates a new action process and passes a high priority score to the process. The action process attempts to move the robot away from the point of collision, which is accomplished by moving the end effector a fixed distance (currently 10cm) in the opposite direction from the collision.

A contact point is reported whenever a torque-based collision or a touch-based contact is detected. This contact point is written to the interaction history for the collision process, and touch-tracking processes for active object schemas (see the next section) can claim the contact point as touch data for their object schemas. Torque-based collisions, which are based only on the knowledge that a motor in the arm is exerting high torque, lead to a contact point based on the assumption that the collision was encountered at the location of the wrist joint. This is because torque-based collisions most often occur when the arm is moving downwards and collides with an object on the table.

A touch-based contact point detected on the inside surface of a fingertip leads to instantiation of a behavior process that executes a grasping motion with low priority score, supported by the curiosity motivation. This is intended to allow the human to give objects to the robot if the robot is not performing any other tasks at the moment.

Touch-Tracking Processes

Touch-tracking processes (or just "touch trackers") are sensory processes responsible for using touch information to update the predicted location of an object. Each object schema has a touch-tracking process, which coordinates touch information associated with an object schema to determine a touch-based location for the object.

When contacts and collisions are reported by the processes described in the previous section, the contact points are evaluated by each touch-tracking process using a distance metric to determine if it applies to its object schema. If the contact point is close enough to the current predicted location of an object, the touch-tracking process will write that contact point to its interaction history as a new predicted location for its object. The contact point will be claimed by at most one touch tracker, the one whose object has the closest distance between its predicted location and the contact point. In the case that the contact point is at or below the table surface level, or otherwise out of range, it is ignored by the touch trackers.

An additional always-on unbound process monitors the fingertip touch sensors to determine if the pattern of touch contacts signals that an object is being grasped by the hand. When a grasp contact is detected, there are two cases:

1. The grasp contact occurred during a grasping behavior being taken towards a known object schema. In this case, the existing object schema's touch tracker, when it notices the signal, will take the current position of the hand and continuously treat it as a source of information for the location of the targeted object. This means that it writes the location information of the hand to its interaction history as a location attribute of its object schema. This continues until the grasp contact ceases.

2. The grasp contact occurred during a grasping behavior with no target schema, or the grasp contact occurred outside of a grasping behavior (such as the human placing an object between the fingers). In this case, a touch tracker is instantiated to continuously use the current position of the hand as an object location, but in this case the tracker remains unbound for a set period of time (about half a second) while checking if the grasp contact continues. This is done to rule out noise and transient object-like contacts. After the time period elapses, a new object schema is generated and the touch-tracking process is bound to the object schema, performing updates to the new object schema's location property until the grasp contact ceases.

Object-Directed Actions

The remaining type of arm- and hand-related behavior process handles object-directed actions. These processes are typically instantiated by plan fragment processes as part of a plan hierarchy, and they compete for activation based on their priority score as assigned by the planner and the primary motivations. When this type of behavior process is activated, it assumes control of the arm and hand motor control modules, and sends messages to the motor control modules to carry out its prescribed action.

The actions available via these behavior processes are:

- Grasping the hand by closing the fingers until contact is detected. The plan fragment that wraps the grasping action also includes a reaction control process (see Section 3.4.7) to prevent finger contact from causing collision avoidance reactions.

- Opening the hand in order to prepare to grasp, or to release a grasp.
- Adjusting the finger spread until the fingers are adjacent to each other, as preparation to tap an object.
- Adjusting the finger spread until the fingers are spread apart, in order to grasp an object.
- Moving the arm to a specified location, such as an object, or a point above an object. When performed as part of the plan fragment to tap an object, a reaction control process stops the arm's motion as soon as touch contact is achieved.

In the current implementation, there are two approaches for grasping that the system must select between for any given object: a grasp from the side, which is more forgiving but risks collision with other objects on the table, and a grasp from above, which is less reliable but avoids dragging cables and the arm into other objects. Figure 4-5 shows the two different grasp approaches.

Each of these behavior processes monitors the relevant control modules to determine whether their assigned action has successfully completed or failed. Success and failure outcomes are written to the interaction histories.

Because the grasp-monitoring process (from the previous section) is always watching for grasping actions and signs of grasping from the touch sensors, actions taken by the object-directed action processes can indirectly lead to additional information about objects, such as grasp- or collision-based locations.

4.7.2 Vision-Related Processes

The behavior processes that relate to vision deal with initial segmentation, visual tracking, categorization, and coordinate transforms.

Initial Segmentation

An always-on, unbound behavior process reads incoming region information from the segmentation module and writes data about regions above a preset size threshold to its interaction history. Just as touch contact information is claimed by touch trackers, visual regions are claimed by visual trackers using a distance metric based on size, color, and position information. Any region that goes unclaimed by the current set of visual tracking processes is assigned to a new visual tracking process.

Visual Trackers

The behavior processes that track visual regions work primarily with the input from the segmentation module. When regions are initially assigned to new visual trackers, they start out in an unbound provisionary mode, waiting for their assigned regions to persist for a preset period of time to factor out noise, shadows, and other transient visual artifacts. After the period of time elapses, if the region continues to persist, the tracking process requests the instantiation of a new object schema in the belief context, and becomes bound to the new schema.

When each new frame is processed by the segmenter, each tracking process attempts to claim a region from the segmenter input based on size, color, and position. When a region is available to the tracking process, it records the region's location to the interaction history. If a tracking process sees no region within a programmed threshold for several seconds, it assumes that its object has left the workspace, and records in its interaction history that its object has an unknown visual position. At all times, the tracking process writes interaction history attributes reflecting the most recent color and size of its region.

When claiming regions, the tracking processes stay aware of arm and hand positions written to the interaction histories, because regions in the area of the arm and hand are often occluded by the robot parts and thus considered untrustworthy. This area is called the "occlusion zone." Unexpected regions in the occlusion zone receive no visual tracking process, because of the possibility of partial occlusion or confusion with regions of the arm. Whenever an object's last known position was in the occlusion zone, and no region outside the occlusion zone can be claimed, the tracking process takes no new data and continues to report data taken before the arm moved near the object. Each element of data reported by a tracking process has a timestamp, so it is clear that the data is not being updated.

The main mode of tracking of regions is done using by comparing regions in a new frame with prior properties of the tracked region. For an object being targeted by a motor action, the mean-shift tracking module's capabilities can additionally be integrated into the tracking of an object. When mean-shift tracking is desired, an additional process is instantiated whose purpose is to communicate with the segmenter and the mean-shift tracker to provide templates from the segmenter to be tracked by the mean-shift tracker. When mean-shift tracking is active for an object, attributes from the mean-shift tracking are also recorded to the interaction history.

The main visual tracking process for each object checks the interaction history to see if mean-shift tracking data is available for its object, and when it is, the mean-shift position data is generally favored over the data from the segmenter for reporting of an object's visual position, depending on the confidence of the mean-shift data and whether the region is in the occlusion zone.

Categorization Processes

The color, shape, and weight ("light" or "heavy") of each object needs to be categorized for both planning and language purposes. Categorization processes are assigned to each object schema for this purpose. The shape process coordinates with the segmenter and the shape recognizer by passing visual regions from the segmenter to the shape recognizer for shape categorization. After the region is passed, the shape process checks periodically for the result from the shape recognizer, which requires a brief processing time. The result of the shape categorization is then written to the shape field in the interaction history as an attribute of the object. The color categorization process is implemented to write color category data directly from the segmenter to the color element in the interaction history, which is trivial because the segmenter makes use of vector-quantized discrete color information to perform its initial segmentation. Likewise, weight is categorized based on a programmed threshold and written appropriately.

Coordinate Transformation

The two coordinate systems used throughout the implemention are the 2-dimensional visual position, reported in pixels relative to the upper-left of the camera's input image, and the 3-dimensional physical location, reported in centimeters relative to the shoulder joint of the robot. Each object schema has a coordinate transformation process associated with it, which takes pixel positions reported by visual tracking processes and transforms them to predicted physical coordinates, and vice versa for physical locations reported by touch-tracking processes.

Because only one camera is currently in use by the implementation for the sake of simplicity, objects not in contact with the robot hand are assumed to be on the plane of the table. Thus, the coordinate transform involves vector arithmetic and camera distortion correction to determine the physical location of an object seen on the table.

Coordinate transformation processes are also used by the body schemas in order to convert the physical coordinates of each piece of the arm and hand into visual coordinates so visual tracking processes can determine their occlusion status. Thus, coordinate transformation processes are the other type of behavior process (besides the coordination process that retrieves updated data, in the "Robot State Processes" section above) involved in body schemas.

4.7.3 Other Motor and Sensory Processes

The remaining motor and sensory behavior processes handle control and sensing of the neck and camera positions. The neck and cameras are controlled to look forward towards the human when the robot is speaking or being spoken to, and to look down at the table when the robot needs to see objects and act upon them. The state of the neck and eyes is written to the interaction histories by dedicated, always-on processes that are the coordination process (and only process) of their body schemas, and the neck and eyes can receive commands via action processes that send appropriate commands when activated.

4.8 Coordination Processes

For every instantiated schema, a coordination process is activated and bound. As described in Chapter 3, the coordination process manages other processes associated with its schema, reconciles conflicts between data coming from the other bound processes, and writes reconciled information to the interaction history.

Of the three types of schemas, only object schemas have substantial functionality in their coordination processes. Body schemas only involve an always-on pair of processes, the coordination process that writes updated data about robot parts and hardware to the interaction history, and a coordinate transform process that transforms coordinates for locations of robot arm parts into visual coordinates. Thus, the coordination process has no additional process management to handle.

Likewise, location schemas involve relatively little functionality in the coordination process. Because some location schemas specify locations relative to landmarks ("to the left of the red one"), the coordination process in such schemas performs the appropriate computation relative to the bound object schemas and writes the calculated location data to the interaction history.

Coordination processes for object schemas have more functions to perform, because object schemas serve as the point of coordination for multiple interaction modalities:

- The coordination process activates shape and color categorization processes as soon as the object schema has an active visual tracking process associated with it. Likewise, the coordination process activates the coordinate transformation process as soon as the object has either an active visual tracking process or an active touch tracking process.

- The coordination process instantiates a mean-shift tracking process on its object whenever it has a visual tracking process. Because the mean-shift tracking module can only handle one object, these processes compete based on the priority scores of activated action-handling processes targeting each object, and only the mean-shift process with highest priority can control the mean-shift tracking module.

- The coordination process deactivates the visual tracker and the mean-shift tracker whenever they report poor confidence (distance metrics) in their tracking. If both tracker types have poor confidence, then the coordination process also deactivates the categorization processes and the coordinate transform processes as well. Previously written information stays in the interaction history, but the timestamp becomes "stale," enabling the coordination process to determine whether touch-based information is newer and thus more reliable.

- The coordination process examines the data written to the interaction history by both types of visual trackers and by the touch tracker associated with its object schema. Based on confidence levels provided by the trackers and based on timestamp information about how recent the data is, the coordination process writes a single reconciled predicted location for its object schema.

- As in Interaction 5 in Chapter 1, if locations provided by visual trackers and touch-based trackers indicate a discrepancy, then the coordination process assumes that the visual trackers were able to stay locked on the visual form of the object, so it unbinds the touch tracker from its schema, allowing the unbound touch tracker to request the creation of a new schema. The new schema will start out with no visual trackers, and thus have unknown visual properties until it is released and the arm moves away from it so it leaves the arm occlusion zone.

4.9 Planning and Talking

The mechanisms of the planning system and the language interaction have already been discussed in Chapter 3. Here I present a simplified list of the plan fragments and condition types used for generating the behaviors in the implemented system, along with verbs used to instantiate them from verbal input.

PickUp(object) is defined as a condition to be holding the object at a point above the table. It can be instantiated with the verbal command "pick up." It is also used by the curiosity motivation to attempt to grasp and lift each object.

AtLocation(object, location) is defined as a condition in which the given object is at the given target location. The definition of `PickUp` actually uses `AtLocation` for its implementation, where the target location is at least 15 centimeters above the table surface.

HandMe(object) is defined as a plan fragment that first requires `PickUp`, and next in its sequence is an action process that moves the arm to a programmed location for handing the object, followed by opening the hand and then retracting the arm.

IsGrasping(object) is defined as a condition to have successfully closed the fingers around the object.

AtObject(object) is defined as a condition to have the open hand near the position of the object.

MoveObjectTo(object, location) is defined as a plan fragment that first requires the target object be held in the robot's hand above the table, and then instantiates an action process that moves the hand to the target location. It achieves the effect of the object being at the target location, `AtLocation(object, location)`, and thus can be used to fulfill the `PickUp` condition and any `AtLocation` condition. For moving the object to another location on the table surface, it can be instantiated with the verbal command "move *ref* to *location*."

MoveToObject(object) is defined as a plan fragment that first requires that the hand be open and empty, and then instantiates an action process that moves the hand to the target object location. It achieves the effect of being at the object's location, thus fulfilling the `AtObject` condition.

Grasp(object) is a plan fragment that requires `AtObject(object)` as a precondition, and runs the action process to close the fingers until contact is detected. The unbound grasping-detection process watches for this plan fragment to determine the identity of an object that is grasped. It can be run unbound, to simply close the hand with no known targeted object.

Tap(object) is defined as a plan fragment that first closes the spread of the fingers, and then performs a `MoveToObject` while activating a reaction control process to react to contact by immediately withdrawing the hand. It can be instantiated with the verbal command "touch *ref*."

Set*X*(object) is defined as a plan fragment that runs a process to revise the interaction history of the object to include the categorical attribute X, where X could be `heavy` or `light` or potentially other values as well, but for the current implementation only the weight is used.

Simple locations are generated from verbal input by taking phrases like "to the left of *referent*" for the prepositions "left," "right," "above," and "below," and creating a location schema whose coordination process translates the location of the object schema by 15 centimeters in the appropriate direction.

4.10 An Example Interaction, Revisited

Having now described in greater detail both the model and the mechanisms of the implementation, I will narrate Interaction 3 from Chapter 1 in more detail. Rereading the examples in Chapter 1 after reading about the model and implementation may also give additional insights into the implemented interactions.

The human adds a red apple to the tabletop and issues the command, "Group the green block and the red apple." Trisk reaches for the red apple, but before grasping it, the human then says, "The red apple is heavy." Trisk immediately backs away from the red apple, and opts to lift and move the green block instead.

As the red apple is added to the tabletop, the visual segmenter produces a corresponding red region and a new visual tracker is assigned to follow the red region. After several frames of consistent segmentation of the red region, the tracker requests the creation of a new object schema. The delay of several frames helps to filter out spurious visual regions. The new object schema, which will be referred to as `Obj1`, is bound to a new coordination process and the visual tracker. The visual tracker writes continuous pixel, color, and visual coordinate attributes to the interaction history of the object schema. The coordination process immediately requests shape and color categorization processes and a coordinate transform process. These translation processes read the attributes written by the visual tracker and write `apple`, `red`, and an arm coordinate attribute to `Obj1`'s interaction history.

A similar procedure takes place for the green block, whose object schema will be referred to as `Obj2`.

The human issues the command, "Group the green block and the red apple." The speech input goes through the speech recognizer and the parser, which produces a parse tree with a verb token for "group" and noun phrase tokens for "the green block" and "the red apple." The language processing system receives the noun phrase tokens and generates reference processes for each one, passing the description tokens for "green" and "block" to reference process `Ref1`, and description tokens for "red" and "apple" to `Ref2`. The language processing system also instantiates the plan fragment `Group(Ref1, Ref2)`, which receives a priority score of 0.6 (hand-coded) from the social motivation.

The plan fragment `Group(Ref1, Ref2)` cannot be used until its reference processes have resolved into objects. `Ref1` and `Ref2` each match their description tokens with the categorical attributes available for `Obj1` and `Obj2`, and resolve to their matched referent schemas. The plan fragment now acts as if it were `Group(Obj2, Obj1)`, and can expand its arguments and activate. The plan fragment starts its sequence with a condition process `Grouped(Obj2, Obj1)`. The condition process inherits the 0.6 priority score from the plan fragment, and the planner generates a list of potential behavior processes from the current list of object schemas. The potential behavior processes whose effects would address condition `Grouped(Obj2, Obj1)` are `MoveObjectTo(Obj2, Obj1)` and `MoveObjectTo(Obj1, Obj2)`.

The planner must now evaluate the success rates for the two `MoveObjectTo` plan fragments. Because neither object schema includes a `weight:heavy` attribute, they have the same expected success rate, and the planner arbitrarily selects `MoveObjectTo(Obj1, Obj2)` to complete the condition. Plan fragment `MoveObjectTo(Obj1, Obj2)` has two elements in its sequence: Condition `IsGrasping(Obj1)`, and action `MoveTo(Obj2)`. The planner attaches `IsGrasping(Obj1)` to `Move(Obj1, Obj2)`, and likewise adds plan fragment `Grasp(Obj1)`, condition `AtObject(Obj1)`, plan fragment `MoveToObject(Obj1)`, and finally action `MoveTo(Obj1)`.

When the action process `MoveTo(Obj1)` inherits the priority score, it activates and sends motor commands, resulting in the arm moving towards the red apple in order to pick it up. At that moment, the human says "The red apple is heavy." This leads to a new reference process `Ref3` to match with `Obj1`, and a new plan fragment `SetHeavy(Ref3)`. This plan fragment gets a priority score of 0.7 from the social motivation, chosen to supercede the plan fragments for physical action because setting an attribute can be done instantly. Once `SetHeavy(Ref3)` has a resolved reference process, it activates its single sensory process, which is to write `weight:heavy` to the interaction history for `Obj1`.

With `weight:heavy` in the interaction history for `Obj1`, the interaction history for plan fragment `MoveObjectTo` now has a different history, because past object schemas with `weight:heavy` attributes have a history of failures. The failures in the history lead to a poor success rate, and `MoveObjectTo(Obj2, Obj1)` now dominates. The planner immediately cuts off the priority score inherited by `MoveObjectTo(Obj1, Obj2)` and attaches `MoveObjectTo(Obj2, Obj1)` to the `Grouped(Obj2, Obj1)` condition instead. Now, the corresponding set of plan fragments and conditions for `Obj2` are attached, and when action `MoveTo(Obj2)` inherits the priority score, it activates and sends its motor commands. The arm moves to the green block. When the hand arrives at the green block, `MoveTo(Obj2)` writes its successful outcome to the interaction history, and exits. Its parent condition, `AtObject(Obj2)`, writes `true` to the interaction history, which causes the parent plan fragment `Grasp(Obj2)` to advance to the next element of its sequence, the action process `CloseHand`. The hand closes, at which point the touch tracking process for `Obj2` detects a successful targeted grasp and begins writing arm coordinate attributes to the interaction history for `Obj2`.

The plan hierarchy continues advancing through plan fragment sequences as each condition yields `true` in turn, and the green block is lifted and moved to a position near the red apple. Once the green block is placed near the red apple, the condition process `Grouped(Obj2, Obj1)` yields `true` and the top-level plan fragment `Group(Obj2, Obj1)` can advance to its next sequence elements. At the end of the sequence for `Group(Obj2, Obj1)` are action processes for the hand to ungrasp and the arm to return to a home position. As the hand ungrasps, the cessation of the grasp is detected and the touch tracking process stops writing arm coordinates. The arm returns to its home position out of view of the camera, and `Group(Obj2, Obj1)` records a successful outcome and elapsed time to the interaction history of both objects. Having completed, the plan fragment no longer receives the priority score from the primary motivation, and it is eventually garbage-collected.

4.11 Summary

This chapter described details of the implementation. Because Chapter 3 focused on the model and the mechanisms of the four perspectives, this chapter focused on presenting the modules that carry out the sensorimotor behaviors of the system, and the specifics of the implementation that generate the current behavior of the system.

Chapter 5

Evaluation

The main direction for this line of research is to build representations suitable for language, and then connect them to language. The primary contribution of this thesis is the Object Schema Model as a means of integrating vision, touch, motor action, planning, and language in a way that supports specific types of language use. As such, an appropriate means of evaluation is to examine the model in terms of the types of language use and representation that it enables, and in doing so to discuss the assumptions and limitations of the model.

5.1 The Sum of the Parts

Language grounding is not a very large field at this point, and each project typically handles such a unique set of linguistic behaviors that comparison would be a matter of apples-and-oranges. Comparing to specific subdomains is not necessarily very informative, either.

As parts of a system focused on integration, the individual components are strong, but their performance is not the goal of the project. As it is, each component compares favorably with other components in their respective domains; in many cases, this is because our components are derived directly from fairly recent work in those domains. For more information, see the respective references on the visual segmenter [16], mean-shift tracking [22], shape contexts [8], our Earley parser [36], or the Sphinx 4 speech recognizer [23].

In these sorts of specific subfields, there exist standard benchmarks. For instance, shape recognition has the COIL dataset [57], action selection has a contest in a simulated domain

[42], and planning has the PDDL competition [32].

However, the contribution of the OSM is the integration of these components in a way that enables language grounding in addition to several novel integrative aspects. The parts have been designed or modified for this integration. For instance, the action selection and plan hierarchies in the model are specifically geared for having primary motivations control behavior processes linked to object schemas, making use of predictive knowledge based on the interaction histories and verbal input.

To evaluate in the model in terms of its separate components would ignore the intent to make the model more than just the sum of its parts. As an example of the value of such integration, tactile and visual inputs taken together clean up noise and reconcile object identities better than data from either single modality alone (and the value of multimodal processing has been shown in other domains as well, e.g., [71]).

Proper evaluation of such a model thus involves exploring the capabilities and limitations of the fully integrated model, rather than focusing on limitations of its current implementation and its individual components.

Therefore, rather than compare my system to benchmarks in specific subdomains or other language-grounding systems with markedly different goals, I will first discuss the OSM in the context of the Token Test [24, 52, 53], a standard test used to assess language and learning deficiencies in children and language-impaired adults. Then, I will discuss some limitations, assumptions, and possible directions for extending the model.

5.2 The Token Test

The Token Test [24, 52, 53] is a standard test for assessing language and learning impairments relative to a sample of "normal" same-aged people. There are multiple variants of the test, each geared for a slightly different group. In this case I will primarily discuss the Token Test for Children [52], and mention other variants when appropriate. The Token Test for Children is administered by placing small circular and square tokens of various colors in front of a child and having the examiner ask the child to perform specific tasks with respect to the tokens. The responses of the child are used to determine the level of the child's language impairment relative to other children of the same age, for children ages 3-12.

The test is divided into four sections. Here are some representative questions from each of the sections:

Section 1 Touch the small green circle. Touch the large yellow circle. Touch the large blue circle.

Section 2 Touch the yellow circle and the red square. Touch the blue square and the white circle. Touch the red square and the white circle.

Section 3 Touch the small yellow circle and the large green square. Touch the large white square and the large red circle. Touch the small blue square and the small yellow circle.

Section 4 Put the white square behind the yellow circle. Touch—with the blue circle—the red square. If there is a black circle, pick up the red square. Put all the squares, one on top of each other, so that in the end the white one is on the bottom and the yellow one is on the top. While touching the red circle, touch the red square, but first touch the green circle.

5.2.1 Relevance

The relevance of the Token Test is that it represents a standard assessment of human language comprehension. Examining the increasing complexity of commands in the Token Test can give some sense of what kinds of conceptualization and language grounding are considered normal in developing children and thus would be suitable for targeting in an artificial language-using system. Thus, comparing the abilities of the OSM with the commands in the Token Test serves as an evaluation of the progress made by the OSM towards some of the first basic levels of human language use, and also provides insight into the structures and processes that might be necessary for the next levels.

Unfortunately, the Token Test in its official form cannot be handled by the current implementation. The most obvious hurdle is that the tokens for the token test are small

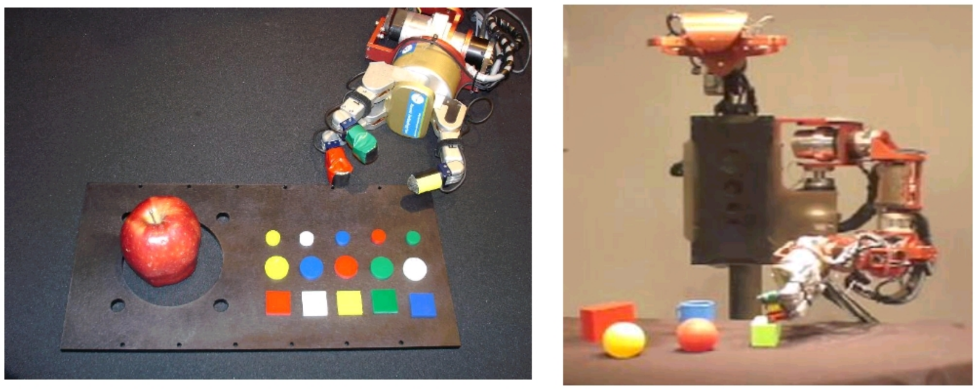

Figure 5-1: The Token Test is a set of commands used for testing the language comprehension level of children and language-impaired adults. Left: The robot hand next to the official tokens used for the test, and an apple to give a sense of scale. Right: Trisk responds to "Touch the green block," using objects more appropriate to its scale.

flat tokens about an inch across. The fingers of the robot hand would be unable to grasp such objects even if carefully handed to it by a human (see Figure 5-1). Instead, we have the robot work with a more limited set of appropriately-sized objects of similar complexity.

The first three sections of the Token Test can be performed straightforwardly by the current implementation. The categorization of colors, shapes, and sizes is already performed by the system, so identifying and touching "the small green circle" is no problem. The use of conjunctions ("Touch the x and the y") connects to the creation of a sequential plan that touches one referent object and then the other. The current implementation is also equipped to handle other simple sequential actions, such as "Before touching the x, touch the y," or "Touch the x after touching the y." Some of these forms are found on other variants of the Token Test. The "touch" command makes use of the Tap plan fragment described in Section 4.9. Figure 5-1 also shows Trisk responding to the command "Touch the green block," in a Token Test-like situation with objects more suitable for its size and dexterity.

Section 4 of the Token Test exposes a much larger number of weaknesses in the current implementation. I will discuss the listed excerpts in order.

- **Put the white square behind the yellow circle.** This form is already implemented in the current model, by connecting a preposition like “behind” to a location schema that computes a target location (“behind”) relative to an object referred to by a reference process (“the yellow circle”). The robot can move an object from its current location to the provided target location.

- **Touch—with the blue circle—the red square.** A proper implementation of this command requires some notion of tool use – carrying out actions using a specific instrument to replace the robot’s fingers, which would be the default instrument. This would require reworking of the model to explicitly represent the instrument used for each action, defaulting to the body schema representing the robot hand. Additionally, the model would need some sense of effector shape, such as the shape of its hand compared to the shape of the current instrument. All the current actions would have to be reworked to make use of the sense of body and the parameterization of instrument. While it would not break the current model’s characteristics, it would take substantial effort to implement.

- **If there is a black circle, pick up the red square.** Hypothetical conditions are not part of the current model, because it requires the representation of a condition that does not necessarily exist in the here-and-now, and then acting based on that condition. The proper way to implement this is to use higher-order situation schemas, in which an entire state of the belief context and of the primary motivations is referred to by a single schema structure. This would enable representation of a hypothetical condition (“if there is a black circle”), which could then be compared to the current situation schema in order to determine whether to take the action or not. This would require such substantial reworking of the model and implementation that possibly only the basic ideas from the current model would be retained. Situation schemas will be discussed further in Section 5.4.

- **Put all the squares, one on top of each other, so that in the end the white one is on the bottom and the yellow one is on the top.** This would require changes to a number of aspects of the implementation. First, to sustain the premise that the model should have sensory contact with its representational referents, the neck needs to be put into action to find different angles of view to check the state of the stack of blocks. Next, the vision system would need the ability to perceive the stacked blocks.

After that, there are two possible approaches. First, new condition process subclasses could be added to handle three-dimensional relations between objects, and plan fragments could be added to enable stacking of objects. The stack of objects could be generated by backchaining a plan to make a certain set of object relations hold. This would stay fairly well within the abilities of the current model.

Alternatively, I believe a more compelling solution would be to represent the stack of blocks as a new level of abstraction. It would be an object schema itself, as well as being composed of other object schemas, with a certain set of relations holding between the sub-schemas. The planner would then need to be extended to handle the arrangement of sub-schemas within the stack schema, including three-dimensional relations.

This second solution stays more in line with the human ability to track only a limited number of objects. Rather than simultaneously tracking and planning on all the objects in the stack, the system should track the stack schema and a few sub-schemas that need to be manipulated relative to the stack. This approach can be implemented as an additional set of conditions, plan fragments, and a hierarchical schema structure without substantially affecting the properties already in the model.

- **While touching the red circle, touch the red square, but first touch the green circle.** Sequencing multiple actions via words like "before," "after," and "first" is already implemented, and leads to multi-action sequential plan fragments as in the case with "and" conjunctions. On the other hand, the robot only has one hand, and adding and coordinating another hand would take a large amount of effort.

5.2.2 Implications

The Token Test is useful because it demonstrates a set of linguistically-specified tasks that are considered simple in human terms, thus providing a benchmark for basic natural language comprehension in a robot system. The current implementation of the OSM can pass a sizable proportion of the commands in the Token Test. A few of the unaddressed commands in section 4 would require major additions to the model but would otherwise leave the already-functional parts of the model intact. The other unaddressed commands, such as those requiring a sense of tool use or hypothetical situations, would require major revisions and in many ways implementation of a substantially different model.

As it stands, the parts of the Token Test that can be passed by the OSM are an encouraging sign for the utility of the integrated model. Because the OSM was designed as a general model of multimodal integration and certain types of language grounding, its abilities towards the Token Test indicates that it is a good first step towards more complex language comprehension.

On the other hand, the lessons learned from the Token Test are not completely straightforward to interpret. The reason the Token Test is difficult for impaired humans is different from why the Token Test is difficult for a general-purpose language grounding system. In the case where a human understands the words used in the Token Test, the human may still be limited by working memory and attentional impairments. In contrast, the OSM is not built with specific limits to the number of instantiable objects and plan fragments. Instead, the OSM simply lacks the full range of conceptual structures necessary to deal with more than simple concrete physical objects in the immediate environment.

Furthermore, the Token Test tests only a small sampling of the total set of behaviors that would be expected of a young child, and thus has nothing to say about many of the other capabilities of the implemented robot system (or of young children). The tasks make sparing use of the grasping and picking-up actions, and, due to being a test of language comprehension, the test ignores responsiveness to external physical events. If a child passed the Token Test but was otherwise unresponsive to its world, it would be cause for alarm.

Other versions of the Token Test do test at least some form of plan revision or interruptibility, such as “Touch the blue square... no, the red square.” The implemented Trisk system can revise plans based on certain types of linguistic correction, including “Touch the blue square... no the red square,” and also “Move the red block to the left... no, to the right.” While this successfully demonstrates the reactivity of the planner to corrections, a more compelling implementation of such interactions would require a more complete sense of dialogue and what it means to participate in a verbal exchange.

As a caveat, the Token Test would be a fairly artificial goal to target for a general-purpose grounded language robot. The OSM was designed to handle multimodal integration and language grounding, and not specifically to pass elements of the Token Test. It is in light of this that I consider the Token Test an acceptable evaluation for the model, in that the Token Test is not the only set of interactions supported by the model. If a system were to be designed specifically to move a robot to pass the Token Test, it could be done with a far more limited representation and not necessarily be all that convincing. While this exposes the Token Test as a poor standard for any sort of "objective" evaluation, it is still a good, accessible standard of evaluation for a grounded language system with a unique feature set.

5.3 Assumptions and Limitations of Object Processing in the OSM

Beyond the Token Test, the OSM and its current implementation on Trisk make a number of assumptions about object processing that may limit its generalizability and applicability to a full range of situations. Apart from the basic perceptual limitations that objects be required to have solid bright colors, have some robot-manipulable size, and exist in the immediate physical environment, there are a few more conceptual-level assumptions being made.

5.3.1 Object Identity Assumptions

In order to produce a working implementation in the specific domain of solid-colored, simple physical objects, certain assumptions had to be made in order to establish and sustain object identities. Each unexpected visual or touch sign leads to a new object schema, and from the moment of its creation, each object schema uses a set of distance metrics hard-coded in the coordination processes and the various trackers to determine whether a visual region or touch contact represents the same object.

This paradigm makes the assumption that each object should be assigned a unique identity based on perceptual distances. It has two major limitations. First, if an object disappears from view and then a sufficiently similar object is brought into view, no provisions are made to permit any retrospective realization that the two objects are actually different. The model is capable of deciding based on subsequent information that an object is not the one that it was expected to be, and it will then revise its identities by detaching processes and creating a new object schema. However, the data assembled during the time of mistaken identity will not be retroactively attributed to the new object schema.

The second limitation of the notion of unique identity is that it leaves little room for *deictic representations*, where objects are represented in terms of their relation to immediate goals. If the model is instructed to manipulate a green block, and, for instance, green block #2 is currently on the table, it necessarily has to target all command-related actions towards what it believes to green block #2. It cannot represent objects solely in terms of their usefulness relative to its current purposes.

On one hand, this precludes representations like "the bee that is chasing me," as in Agre and Chapman's use of deictic representations [1], where the unique identity of the bee is not important, because the system is concerned about any given bee that it identifies as chasing it. On the other hand, language referents typically do refer to specific unique identities. "Pick up the green block" refers to a specific green block, presumably the green block that is present at the moment of the utterance. If a new green block comes into view, the utterance does not also refer to the new green block, even if they are otherwise identical. Thus, the lack of deictic representations is a suitable assumption for language use, but it would be useful to include deictic representations for handling specific goal-related behaviors with generic object references. Ostensibly, this could be added to the model by using reference processes as targets of built-in plan fragment subclasses, rather than only for connecting noun phrases to referents.

5.3.2 Object Cohesiveness Assumptions

The model is currently also ill-suited to handle objects that come apart or merge together. While the OSM can handle visual and tactile discrepancies by selecting one modality to trust based on hard-coded trust levels, when the discrepancy is due to an object falling apart or merging, there is no provision for representing either that a wholly new object is present, or for treating the new object in relation to its previous state.

The proper solution to this assumption is related to the hierarchical schemas mentioned in the section on the Token Test task of forming a stack of objects. There would need to be some sort of schema hierarchy, in which a schema can be composed of other schemas. This would be a significant change to the model.

It is worth noting that the construction of schema hierarchies would be a suitable substitute for reference processes. Rather than creating a special structure to represent a noun phrase, an object schema could be created instead, representing an object with the categories described by the noun phrase, and then this new object, which only has a verbal extent, could then be merged with an object schema with physical extent based on feature similarity. In this way, object schema merging does not have to take place only in one modality, or only in the physical sense.

5.4 Future Directions

In this section, I describe a few additions to the model that would fit within the model in its current state, and then I expand the discussion to explain how the model fits into a larger path towards robots capable of general-purpose language use.

At this point, work is ongoing to add more functionality to the reference processes used in verbal interaction. Elements of noun phrases such as direct and indirect articles, or singular and plural referents, can be added by modifying the search mechanism employed by the reference processes and extending the behavior processes involved in the plan hierarchies to accept multiple arguments.

Work is also ongoing to add the notion of containment to the model. As a precursor to concepts like tool use, the idea is to enable one object's location to be constrained to the location of its container object. This involves work on the perceptual end, to determine either by attempting to move objects, or by adjusting the neck position, whether one object sits entirely within or atop another. It also requires adding plan fragments whose notated effects include moving a contained object.

The remainder of this section will describe two types of long-term directions for further development of this model.

5.4.1 Types of Learning

At this point, the compiling of interaction histories relative to attributes constitutes a means of learning object affordances in terms of the robot's actions. Additional types of learning would add to the generalizability of the model. Four key types of learning to add would be parameter learning, word learning, reinforcement learning for the motivations, and learning of new actions and effects.

The current Trisk implementation requires the manual setting of a large number of parameters, including distance metrics, collision thresholds, and coordinate transform parameters. Most of these could be learned or adjusted dynamically given some evaluation metric for the learned parameters.

Word learning for object features and actions could be accomplished with a co-occurrence matching algorithm like the one in [66]. If the use of unfamiliar words could be matched with the set of attentionally-focused objects, then it should be possible to find co-occurrences of words and object features over time.

Doing this flexibly would require the addition of an attentional mechanism that could be directed based on human interaction, for instance by the use of gaze detection and pointing, by methods such as those in [72, 73], or verbal reference. Otherwise, the system would be limited to learning labels with the assumption that the human would always label just the object that the robot already happened to be interacting with.

Reinforcement learning could be applied in order to determine how actions in various contexts lead to changes in the primary motivations. This could be accomplished using a saliency mechanism to determine whether a significant change had recently occurred in the priority levels of one of the primary motivations, and then noting the actions that seemed to contribute to the change. This would then affect the priority scores given to actions by the primary motivations. The C4 system developed by Blumberg et al. [19, 44] makes use of reinforcement learning to learn the motivation effects of action tuples, which then alters the priority with which those actions are selected in service of each motivation. Such an approach would complement the object-centered effect probabilities already being compiled by the OSM.

Finally, learning of new actions and effects could be accomplished by enabling the system to learn the way humans do: motor babbling, direct instruction, and imitation. Motor babbling could enable the system (perhaps in simulation for safety reasons) to try random actions to see how they affect the primary motivations and the state of various conditions bound to objects. A saliency filter would help with identifying conditions and attributes that changed significantly as the result of an action. Actions with salient results could be recorded in order to identify patterns over time.

Likewise, direct instruction by manually moving the robot arm through an action could be a way to add new tasks to the robot's vocabulary. Unpublished results on the Ripley system demonstrated that a Hidden Markov Model representation had potential for direct instruction of tasks like simple grasps. Again, a saliency filter would enable the system to learn about the effects of a learned action.

Imitation would be another suitable method of teaching the robot new actions. It would require improvements to the vision system to enable pose estimation of the human and a "mirror" system capable of translating the pose into the robot's body frame. Some work in that direction has been done by other groups [2, 10, 13].

Situation Schemas

Humans typically consider possible actions based on affordances of current objects and the current task [51]. Thus, it makes sense for the model to consider only the set of actions relevant in the given belief context, rather than searching through all possible plan fragments. This would improve scalability and enable a number of additional representations for language connection and planning.

Thus, a particularly useful abstraction level would be the level of situations, in which the set of objects in the belief context, the relations between objects, the state of the primary motivations, and elements of social context, such as the desires of the person, are organized into a new type of schema that can then be used to connect to words describing the situation or event, as well as serve as discrete semi-stable contexts for determining a set of plan fragments to search to build the plan hierarchy.

As mentioned in the Token Test section, situation schemas would be useful for representing hypothetical situations that could then be used to ground conditional action. It might also add the ability to talk about actions and plans within specific conditions. By grounding explanations for actions and motivations relative to a situation schema, it would be meaningful to explain to the robot the appropriate behaviors and motivations to take within specific situations, such as games, roles, or future situations. Situation schemas would then provide a means of organizing states of the entire belief context and motivation system throughout time.

Work done in Cynthia Breazeal's group [13] constitutes a certain level of situation abstraction. There, the situation abstraction leads to the representation of the beliefs and goal structures of other entities. I suggest that extending this sort of abstraction would enable more complex forms of language grounding as well.

5.5 Overall Assessment

In summary, evaluation according to standard metrics is not entirely useful for a system whose primary contribution is the integration of features not previously integrated.

The discussion of the Token Test is intended to describe the system in terms of progress towards human-level language comprehension, and I believe the OSM does show promise and ability in such a direction. I do not, however, believe that the Token Test should be used as a goal for language grounding experiments, but it does make an interesting progress indicator for work already motivated by the goal of general-purpose representations for language grounding.

Some assumptions made in the current model lead to limitations in the generalizability of the model, but these were primarily made in order to complete the current implementation to handle language about simple objects in the immediate environment. I believe the model can be supplemented to improve its flexibility with respect to deictic representations and hierarchical object structures.

Ongoing work shows the model facilitating the development of more complex verbal interactions and the use of notions like containment. The addition of learning and situation-level abstractions would be good long-term developments for such a model.

Chapter 6

Conclusion

The main purpose of this thesis is to provide the responsiveness of behavior-based systems in a system that generates semi-stable representations for planning and grounded language use. To this end, I have described the Object Schema Model (OSM), which makes use of behavior processes organized into object schemas for object representation. The four perspectives on the OSM demonstrate various means of organizing and manipulating behavior processes to achieve the goals of the system.

- The behavior processes:
  - run concurrently, like thread objects.
  - handle sensing, action, and data management.
  - write data to appropriate interaction histories for use in planning, language processing, and other behavior processes.
- The belief context perspective:
  - organizes behavior processes into schemas, and stores an interaction history for each schema.
  - The schemas and their interaction histories represent a discrete, semi-stable set of symbols for planning and language use.

  - Because schemas are made of behavior processes, they represent objects in terms of the behaviors taken towards them. They also enable integration between touch and vision for reconciling object identities and attributes. Because they are the result of constantly-active sensorimotor processes, they permit rapid response to changes in the environment.

- The planning perspective:

  - organizes behavior processes, especially condition processes and plan fragment processes, into plan hierarchies.
  - One plan hierarchy is active and being constructed at a time, by selecting the plan-forming process with the highest priority score.
  - Priority scores stem from the primary motivations, such as curiosity, collision, and social motivation, which examine the state of the belief context in order to decide on behavior processes to support.
  - The construction of the plan hierarchies is informed by the data in the interaction histories, by favoring actions with higher reliabilities.
  - The plan hierarchies are rebuilt rapidly in response to changes in the belief context and the interaction histories.
  - Actions that receive priority scores compete based on priority score to control necessary resources.

- The language perspective:

  - connects noun phrase tokens and their constituent description tokens to reference processes, which serve as stand-ins for binding schemas to behavior processes until a referent is matched.
  - connects verb tokens to plan fragment processes, allowing the social motivation to provide a priority score and initiate plan hierarchy construction.

6.1 Contributions

As mentioned in Chapter 5, the strength of the OSM and its implementation is in the integration of the various modalities and components. I believe that the path towards success in the goals of artificial intelligence needs to involve more integration, rather than focusing on isolated improvements.

As such, the main contribution of this thesis is the use of semi-stable object schemas and their behavior processes and interaction histories. The use of schemas permits integration between responsive sensorimotor behaviors on one hand, and planning and language use on the other. Because all four perspectives of the system amount to different ways to organize behavior processes, changes in one perspective rapidly affect structures in other perspectives, making for a more interdependent integration than merely stacking completely separable modules.

Within the main contribution of this integrated model, several secondary contributions are made:

- The object schemas permit integration of multiple sensorimotor modalities, with vision providing expectations for grasping actions, and tactile contact providing expectations for vision. The use of multiple modalities improves robustness to sensor noise, and discrepancies between modalities can be resolved by adjusting the behavior processes that compose the object schema's identity.

- Similarly, another contribution is the integration of planning, action selection, and high-level language grounding in a physically embodied robotic system. Adjustments at the object level due to environmental changes or due to verbal input can be used for opportunistic planning. The state of the planner can in turn adjust the reactive elements (such as collision avoidance) of the system.

- Object attributes are used as not only perceptual characteristics and structures for connecting words, but can also influence the prediction of action outcomes and planning. This step beyond other language-grounding projects adds a sense of object attributes being useful for action in the environment, not just object identity tracking and language.

- The object schemas are made of behavior processes that represent both perceptual trackers and plan fragments. This provides a single structure for organizing the connection between perceptual attributes and outcomes. It also provides a single structure for grounding terms related to object affordances, like “liftable” or “graspable.”

- Relative to our group’s previous work on the Ripley system (see Section 2.5), an important addition is the ability to detect failure and have the failure affect the decisions of the system in a patterned way. This stems from the integration of the planner and the outcome data.

6.2 Active Perception and the Robot’s Perspective

Using behavior processes as components of object schemas has conceptual merit beyond enabling integrated systems.

One of the key features of the model is its treatment of all perception processes as active behaviors that involve prediction and action. Rather than extracting a model of objects from a static visual frame, visual tracking of an object makes use of a prediction of the object’s location based on history information (including touch contact data), and is thus modeled as a continuous behavior that assesses new data and produces constant updates. Likewise, grasping actions make use of predictions from both vision and previous tactile contact, and update those predictions in light of successes and failures.

This is important for the responsiveness and integration of the system. Also, it centers the representation of objects around the behaviors that the robot can perform towards objects, including perceptual behaviors, and the results that the robot could achieve using those behaviors.

Essentially, it conceptualizes objects from the robot’s perspective, in terms of both what the robot perceives and what it can do. This differs from using a separated representation that represents objects in terms of their perceptual properties while handling actions and planning in a separate module. This idea of representation from the robot’s perspective, in terms of the robot’s behaviors and results, allows language grounding to be better connected to the embodiment of the robot, bringing language grounding more in line with some of the goals of behavior-based robotics. It also views objects in terms of what they afford the robot, moving beyond the view that an object is predominantly a sensory construct.

6.3 Final Words

The Object Schema Model is designed from the ground up to be a model that integrates its various components as interconnectedly as possible. I have discussed several benefits that derive from the diverse ways the various parts interact.

I believe further progress towards helpful conversational real-world machines requires similarly designing for complete integration of all relevant parts. I believe that considerations of integration need to occur early in the design process, before attempting to achieve perfect sensorimotor processing. Even 99% speech or visual recognition accuracy does not guarantee compatibility with an integrated system, and there is a possibility that a certain level of sensorimotor proficiency might only occur within an integrated system.

Bibliography

[1] P. Agre and D. Chapman. Pengi: An implementation of a theory of activity. In *Proceedings of AAAI-87*, pages 268–272, 1987.

[2] R. Amit and M. Mataric. Learning movement sequences from demonstration. *Int. Conf. on Development and Learning*, 2002.

[3] D. Bailey. *When push comes to shove: A computational model of the role of motor control in the acquisition of action verbs.* PhD thesis, Computer science division, EECS Department, University of California at Berkeley, 1997.

[4] D. Bailey, J. Feldman, S. Narayanan, and G. Lakoff. Embodied lexical development. In *Proceedings of the Nineteenth Annual Meeting of the Cognitive Science Society.* Erlbaum, 1997.

[5] D. H. Ballard, M. M. Hayhoe, P. K. Pook, and R. P. N. Rao. Deictic codes for the embodiment of cognition. *Behavioral and Brain Sciences*, 20:723–767, 1997.

[6] C. Bauckhage, J. Fritsch, K. Rohlfing, S. Wachsmuth, and G. Sagerer. Evaluating integrated speech and image understanding. In *Proceedings of the IEEE International Conference on Multimodal Interfaces (ICMI)*, pages 9–14, 2002.

[7] P. Beeson, M. MacMahon, J. Modayil, A. Murarka, B. Kuipers, and B. Stankiewicz. Integrating multiple representations of spatial knowledge for mapping, navigation, and communication. In *Proceedings of the AAAI 2007 Spring Symposium on Control Mechanisms for Spatial Knowledge Processing in Cognitive / Intelligent Systems*, 2007.

[8] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(24):509–522, 2002.

[9] M. Berlin, J. Gray, A. L. Thomaz, and C. Breazeal. Perspective taking: An organizing principle for learning in human-robot interaction. In *Proceedings of AAAI 2006*, 2006.

[10] A. Billard and M. J. Mataric. Learning human arm movements by imitation: Evaluation of a biologically inspired connectionist architecture. *Robotics and Autonomous Systems*, 37(2-3):145–160, 2001.

[11] B. M. Blumberg. Action-selection in Hamsterdam: Lessons from ethology. In D. Cliff, P. Husbands, J.-A. Meyer, and S. W. Wilson, editors, *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*. MIT Press, 1994.

[12] M. C. Bosse, P. M. Newman, J. J. Leonard, and S. Teller. SLAM in large-scale cyclic environments using the Atlas framework. *International Journal of Robotics Research*, 23(12):1113–1139, December 2004.

[13] C. Breazeal, M. Berlin, A. Brooks, J. Gray, and A. L. Thomaz. Using perspective taking to learn from ambiguous demonstrations. *Journal of Robotics and Autonomous Systems Special Issue on Robot Programming by Demonstration*, 54(5), 2006.

[14] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2:14–23, 1986.

[15] R. A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47:139–160, 1991.

[16] J. Bruce, T. Balch, and M. Veloso. Fast and inexpensive color image segmentation for interactive robots. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2000.

[17] J. J. Bryson. *Intelligence by Design: Principles of Modularity and Coordination for Engineering Complex Adaptive Agents*. PhD thesis, MIT, Department of EECS, Cambridge, MA, June 2001. AI Technical Report 2001-003.

[18] J. J. Bryson and L. A. Stein. Modularity and design in reactive intelligence. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1115–1120, Seattle, August 2001. Morgan Kaufmann.

[19] R. Burke, D. Isla, M. Downie, Y. Ivanov, and B. Blumberg. Creature smarts: The art and architecture of a virtual brain. In *Proceedings of the Game Developers Conference*, pages 147–166, 2001.

[20] D. Chapman. Planning for conjunctive goals. Technical Report 802, MIT AI Laboratory, 1985.

[21] Z. Chen, J. Samarabandu, and R. Rodrigo. Recent advances in simultaneous localization and map-building using computer vision. *Advanced Robotics*, 21(34):233 265, 2007.

[22] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5), May 2002.

[23] Carnegie Mellon University, Sun Microsystems Laboratories, Mitsubishi Electric Research Laboratories. Sphinx 4 Java Speech Recognizer. http://www.speech.cs.cmu.edu/sphinx/twiki/bin/view/Sphinx4/WebHome, 2004.

[24] E. De Renzi and L. A. Vignolo. The token test: A sensitive test to detect receptive disturbances in aphasics. *Brain*, 85(4):665–678, 1962.

[25] G. Drescher. *Made-Up Minds: A Constructivist Approach to Artificial Intelligence*. MIT Press, 1991.

[26] G. Drescher. The schema mechanism: constructivist A.I. In S. J. Hanson, W. Remmele, and R. L. Rivest, editors, *Machine Learning: From Theory to Applications*, volume 661 of *Lecture Notes in Computer Science*. Springer, 1993.

[27] R. Duda and P. Hart. *Pattern Classification and Scene Analysis.* John Wiley and Sons, 1973.

[28] P. Fitzpatrick. *From first contact to close encounters: a developmentally deep perceptual system for a humanoid robot.* PhD thesis, Massachusetts Institute of Technology, 2003.

[29] P. Fitzpatrick and G. Metta. Grounding vision through experimental manipulation. *Philosophical Transactions of the Royal Society: Mathematical, Physical, and Engineering Sciences*, 361(1811):2165–2185, 2003.

[30] E. Gat. Integrating planning and reaction in a heterogeneous asynchronous architecture for controlling mobile robots. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI)*, 1992.

[31] E. Gat. Three-layer architectures. In D. Krotenkamp, R. P. Bannasso, and R. Murphy, editors, *Artificial Intelligence and Mobile Robots.* AAAI Press, 1998.

[32] A. Gerevini and D. Long. Plan constraints and preferences in PDDL3. Technical Report RT 2005-08-47, Dept. of Electronics for Automation, University of Brescia, Italy, 2005.

[33] J. J. Gibson. *The Ecological Approach to Visual Perception.* Erlbaum, 1979.

[34] A. M. Glenberg. What memory is for. *Behavioral and Brain Sciences*, 20(1):1–55, 1997.

[35] A. M. Glenberg and M. P. Kaschak. Grounding language in action. *Psychonomic Bulletin and Review*, 9(3):558–565, September 2002.

[36] P. Gorniak and D. Roy. Grounded semantic composition for visual scenes. *Journal of Artificial Intelligence Research*, 21:429–470, 2004.

[37] P. Gorniak and D. Roy. Situated language understanding as filtering perceived affordances. *Cognitive Science*, in press.

[38] J. Gray, M. Berlin, and C. Breazeal. Intention recognition with divergent beliefs for collaborative robots. In *Proceedings of the AISB (Artificial Intelligence and Simulation of Behavior) Convention*, 2007.

[39] K. Hsiao, P. Gorniak, and D. Roy. NetP: A network API for building heterogeneous modular intelligent systems. In *Proceedings of AAAI 2005 Workshop in Modular Construction of Human-Like Intelligence*, 2005.

[40] K. Hsiao, N. Mavridis, and D. Roy. Coupling perception and simulation: steps towards conversational robotics. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 928–933, 2003.

[41] K. Hsiao and D. Roy. A habit system for an interactive robot. In *AAAI Fall Symposium: From Reactive to Anticipatory Cognitive Embodied Systems*, 2005.

[42] M. Humphrys and C. O'Leary. Constructing complex minds through multiple authors. In *Proceedings of the Seventh International Conference on Simulation of Adaptive Behavior*, pages 3–12, 2002.

[43] D. Isla and B. Blumberg. Object persistence for synthetic characters. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS 2002*, 2002.

[44] D. Isla, R. Burke, M. Downie, and B. Blumberg. A layered brain architecture for synthetic creatures. In *Proceedings of IJCAI*, 2001.

[45] M. P. Kaschak and A. M. Glenberg. Constructing meaning: The role of affordances and grammatical constructions in sentence comprehension. *Journal of Memory and Language*, 43(3):508–529, October 2000.

[46] B. Kuipers. An intellectual history of the spatial semantic hierarchy. In M. Jefferies and A. Yeap, editors, *Robot and Cognitive Approaches to Spatial Mapping.* Springer Verlag, 2007.

[47] B. Kuipers, J. Modayil, P. Beeson, M. MacMahon, and F. Savelli. Local metrical and global topological maps in the hybrid spatial semantic hierarchy. In *IEEE International Conference on Robotics and Automation (ICRA-04)*, 2004.

[48] L. S. Lopes and J. H. Connell. Semisentient robots: Routes to integrated intelligence. *IEEE Intelligent Systems*, 16:10–14, 2001.

[49] L. S. Lopes and A. Teixeira. Human-robot interaction through spoken language dialogue. In *Proceedings of the 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2000.

[50] P. Maes. A bottom-up mechanism for behavior selection in an artificial creature. In J. A. Meyer and S. W. Wilson, editors, *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*, 1991.

[51] N. Maier. Reasoning in humans II: The solution of a problem and its appearance in consciousness. *Journal of Comparative Psychology*, 12:181–194, 1931.

[52] R. L. McGhee, D. J. Ehrler, and F. DiSimoni. *The Token Test for Children, Second Edition*. Pro-Ed, Inc., 2007.

[53] M. R. McNeil and T. E. Prescott. *Revised Token Test*. Pro-Ed, Inc., 1978.

[54] M. Minsky. *Society of Mind*. Simon and Schuster, 1986.

[55] J. Modayil and B. Kuipers. Where do actions come from? autonomous robot learning of objects and actions. In *Proceedings of the AAAI 2007 Spring Symposium on Control Mechanisms for Spatial Knowledge Processing in Cognitive / Intelligent Systems*, 2007.

[56] S. Narayanan. *KARMA: Knowledge-based active representations for metaphor and aspect*. PhD thesis, University of California Berkeley, 1997.

[57] S. A. Nene, S. K. Nayar, and H. Murase. Columbia object image library (COIL-100). Technical Report CUCS-006-96, Columbia University, 1996.

[58] N. J. Nilsson. Shakey the robot. Technical Report 323, AI Center, SRI International, 1984.

[59] J. Orkin. Agent architecture considerations for real-time planning in games. In *Proceedings of the Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2005.

[60] J. Orkin. 3 states and a plan: The AI of F.E.A.R. In *Proceedings of the Game Developers Conference*, 2006.

[61] J. Piaget. *The Construction of Reality in the Child.* Basic Books, 1955.

[62] Z. W. Pylyshyn. Situating vision in the world. *Trends in Cognitive Sciences*, 4(4):197–207, May 2000.

[63] T. Regier and L. Carlson. Grounding spatial language in perception: An empirical and computational investigation. *Journal of Experimental Psychology*, 130(2):273–298, 2001.

[64] D. Roy. Integration of speech and vision using mutual information. In *Int. Conf. Acoustics, Speech, and Processing*, 2000.

[65] D. Roy. Learning words and syntax for a visual description task. *Computer Speech and Language*, 16(3), 2002.

[66] D. Roy. A trainable visually-grounded spoken language generation system. *Proceedings of the International Conference of Spoken Language Processing*, 2002.

[67] D. Roy. Grounded spoken language acquisition: Experiments in word learning. *IEEE Transactions on Multimedia*, 5(2):197–209, 2003.

[68] D. Roy. Grounding words in perception and action: computational insights. *Trends in Cognitive Science*, 9(8):389–396, 2005.

[69] D. Roy. Semiotic schemas: A framework for grounding language in action and perception. *Artificial Intelligence*, 167(1-2):170–205, 2005.

[70] D. Roy, K. Hsiao, and N. Mavridis. Mental imagery for a conversational robot. *IEEE Transactions on Systems, Man, and Cybernetics*, 34:1374–1383, 2004.

[71] D. Roy and N. Mukherjee. Towards situated speech understanding: Visual context priming of language models. *Computer Speech and Language*, 19(2):227–248, 2005.

[72] B. Scassellati. *Foundations of theory of mind for a humanoid robot.* PhD thesis, MIT, 2000.

[73] B. Scassellati. Theory of mind for a humanoid robot. *Autonomous Robots*, 12:13–24, 2002.

[74] J. M. Siskind. Grounding the lexical semantics of verbs in visual perception using force dynamics and event logic. *Journal of Artificial Intelligence Research*, 15:31–90, August 2001.

[75] M. Skubic, D. Perzanowski, S. Blisard, A. Schultz, , W. Adams, M. Bugajska, and D. Brock. Spatial language for human-robot dialogs. *IEEE Transactions on SMC Part C, Special Issue on Human-Robot Interaction*, 34(2):154–167, May 2004.

[76] B. C. Smith. *On the Origin of Objects.* Bradford Books, 1996.

[77] A. Stoytchev. Behavior-grounded representation of tool affordances. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, 2005.

[78] T. Tyrrell. The use of hierarchies for action selection. *Adaptive Behavior*, 1(4):387–420, 1993.

[79] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 511–518, 2001.

[80] T. Winograd. *Understanding Natural Language.* Academic Press, 1972.

[81] T. Winograd and F. Flores. *Understanding Computers and Cognition.* Addison Wesley, 1986.

[82] C. Yu and D. H. Ballard. A multimodal learning interface for grounding spoken language in sensory perceptions. *ACM Transactions on Applied Perceptions*, 1(1):5780, July 2004.