Technical Report

1. Research Summary

1.1 Cognitive Architecture Development

The primary technical objective of our Phase II effort was the realization of a cognitive architecture (CA) controlling the behavior of our robotic scrub technician, Penelope™. As shown in Figure 1, Penelope's CA is the central component controlling the mapping of the robot's sensory inputs to its actuator outputs. The system's physical sensors – the digital cameras, microphone, and so on – act as input to the CA while the physical actuators – the motors that move the robotic arm, the voice synthesis system, and so on – are the CA's output. The CA is an artificial intelligence package comprised of a set of *assertions* codifying the system's current state information and a set of *if-then rules* that define the behavior of the robot.

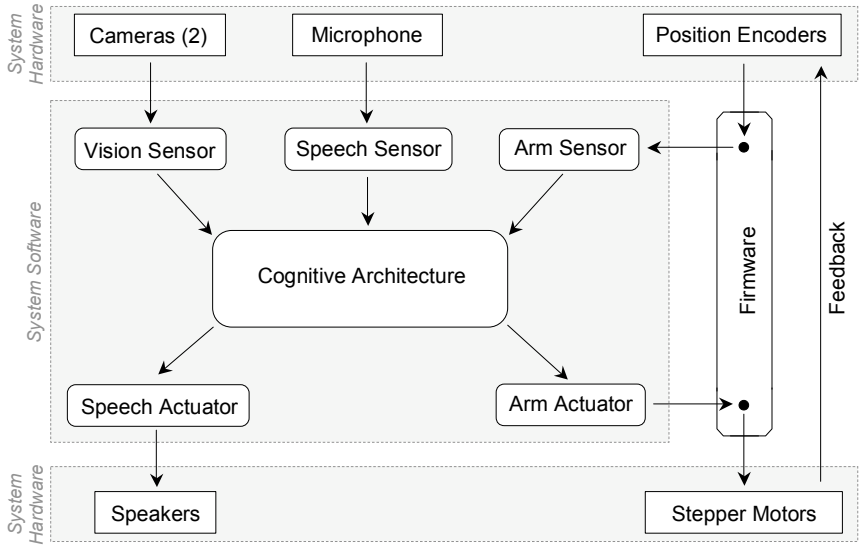

Figure 1. Penelope System Architecture

An example will illustrate how the CA works. If the microphone senor detects a verbal instrument request from the surgeon, that information is packaged as an assertion and presented to the CA. There, a rule states that if there is such a request and the instrument is available on one of the instrument trays, a goal should be created to deliver that instrument to the surgeon. Another rule states that if there is a goal to deliver an instrument, the robotic arm should be moved over that instrument, the electromagnet should be turned on, and so on. In this manner the rules define the actions the robot will take in response to stimuli in its environment. Figure 2 shows some example rules involving these instrument requests.

The rules are divided into rule sets as shown in Figure 3 below. This organization is important from a validation perspective in that it allows us to divide the process into largely independent modules that can be tested both individually and in combination with the other modules.

```
(defineRule
  :description (format nil "Express meaningful voice sensor phrase, ~S,
                            as sensor object ID." sensor.phrase)
  :trigger (and (matchAssertion voiceRecognitionSensor sensor :phrase ~NotNilMatcher())
                (matchFirst commands nil :spokenNames ~ContainsMatcher(sensor.phrase)))
  :action (assertInto voiceSensorObjectID :phrase sensor.phrase))

(defineRule
  :description (format nil "Create a goal to deliver ~A." command.writtenName)
  :trigger (and (matchFirst commands cmdd :command :instrumentRequest
                            :spokenNames ~ContainsMatcher(voiceSensorObjectID.phrase))
                (matchFirst blobs blob
                            :identification cmd.writtenName :surface :instrumentTray))
  :action (assert goals :goal :instrument.delivery :object cmd.writtenName :blob blob))
```

Figure 2. Example rules expressing a goal to deliver an instrument.

In general, the flow moves from sensor expression to goal expression to goal actualization to step execution. There are several rules to accomplish each of these phases. The action of the rules in each phase acts as a trigger for the next phase rules. So sensor expression actions will trigger goal expression rules, and so on. We will now discuss each phase briefly.

Sensor expression refers to the consolidation of raw sensory input into a higher-level form more meaningful to the CA. For example one type of sensor expression might take image data from the cameras and "express" it in terms of the location of the viewed object, it's orientation, and

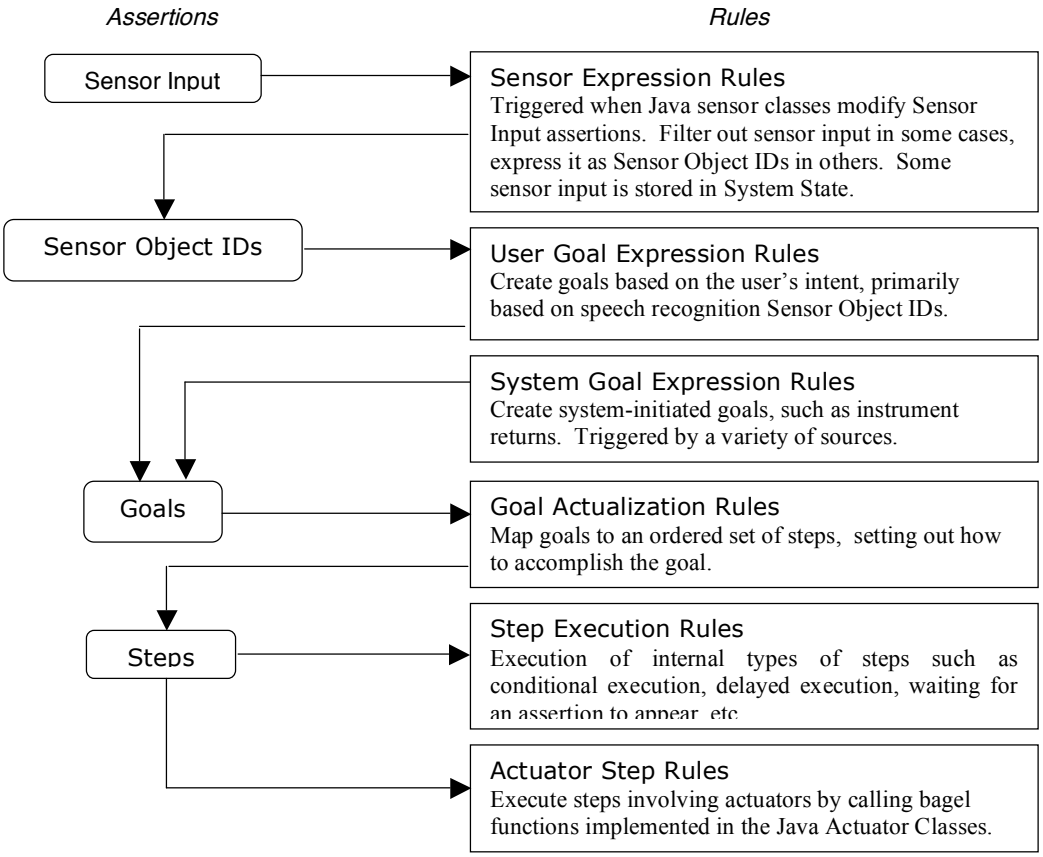

Figure 3. CA Assertion Sets and Rule Sets

any identification information that may have matched from the vision training data. It's a long process from the camera's raw pixel data to this high level characterization. Sensor expression rules are responsible for formatting only the essential information from this process for use in subsequent rules.

Sensor expression rules filter not only the amount of information as we've discussed, but also the frequency. Many of our sensors can provide their input at a high rate. The cameras for example run at 15 frames a second. We don't want to bombard the CA with sensory input, so we use the sensor expression rules to provide the input at meaningful times. The vision sensor expression rules for example wait for motion to cease before analyzing the scene and presenting the new vision sensor input.

The next phase is goal expression. These rules decide if any action should be taken and, if so, they express a goal to accomplish that action. A goal expresses what is to be accomplished, like that an instrument should be delivered. Goals are divided into user-initiated goals and system initiated goals. The canonical example of a user-initiated goal is a goal to deliver an instrument to the surgeon. This goal is a direct result of user input. The analogous system initiated goal is a goal to return an instrument that has been lying on the transfer zone for a significant amount of time. This goal isn't based on any direct user input, but rather a system-initiated desire to keep the transfer zone clean.

Goal expression is usually triggered by sensory input, such as a voice recognition input naming a desired surgical instrument. But there may be several possible triggers for a goal's expression. For example, a prediction of the next needed instrument might trigger an instrument delivery goal instead of voice recognition. This is why we distinguish between sensor expression and goal expression. It provides a well-defined interface between the two concepts and allows the two modules to act independently.

Goal actualization is the next phase. Goal actualization rules are triggered by goal expression. They construct a list of things to do to accomplish the goal. If the goal is to deliver an instrument, this list would include steps to move the arm over the instrument, to turn on the magnet, to then move the arm to the drop-off location, and so on. We have constructed a rich language of step types to support sophisticated goal actualizations. These are essentially programs that the goal actualization rules write. Executing these programs will accomplish the goal. Goal actualization rules also often include a corresponding goal preemption actualization. This is a "program" to be run should the goal be cancelled or otherwise preempted in midexecution. The instrument delivery goal preemption actualization, for example, will check to the see if the arm is currently gripping an instrument. If so, it will generate the steps required to lay that instrument down on the transfer zone as part of the cancellation clean up.

Goal actualizations are a list of "steps". The step execution rules are responsible for implementing each type of step. Some step execution rules initiate actuator output, such as moving the arm, speaking through speech synthesis, and even turning on LEDs. Other rules support conditional step execution, jumps within the step program, delays, and other useful programmatic tools.

1.2 Phase II Objectives

Our Phase II technical objectives were broken down into 3 top level tasks. The following sections describe each task and the extent to which each objective was met. The appendices to this report include other supporting information and test data related to the fulfillment of each of these tasks.

1.2.1 Task 1: Design and Verification of a Cognitive Architecture

> *Objective: The specific deliverable is a working cognitive architecture as described above and a set of working production rules. Unit testing will be done to verify that each production rule produces the desired output.*

This objective was completed successfully. We have developed and thoroughly tested our cognitive architecture. As anticipated in our Phase II proposal, our completed CA enables the robot to reason about its environment in a way that constitutes situational awareness in the OR. Our current rule set consists of approximately 65 rules. Interestingly, the rich capability of our CA has allowed us to abstract *out* of our core Java codebase all concepts relating to the specific job of Penelope – serving surgical instruments. None of the behavior related to identifying, counting, delivering, or returning surgical instruments is hardcoded into our system. Instead, all of this expertise is encoded in our rule set.

Our verification process for the CA has included rule-by-rule unit testing and core CA component testing. We have tested each rule in isolation by defining the system input requirements for the rule's trigger and the system state changes caused by the rule's action. We can then adjust the system inputs to test for proper triggering and introspect the system state knowledge base to verify the action. Figure 4 shows the extensive graphical user interface we have developed to support this. The interface allows us to adjust and monitor any aspect of the system state knowledge base.

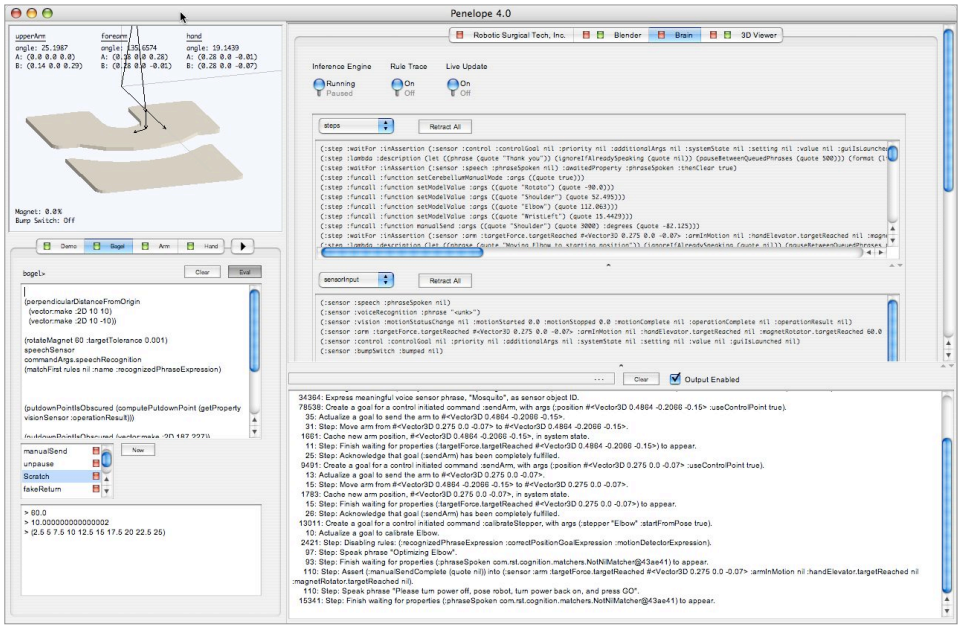

Figure 4. CA Verification and Validation Graphical User Interface

We also developed a custom designed scripting language, called bagel™, in which much of our cognitive architecture is implemented. bagel is lightweight yet extensible Lisp-like scripting language written in and easily integrated with Java. This extra language level allows us to introspect and interact with all of our CA's data structures at run time. This is an essential capability for verification. In bagel we can check the status of all assertions and rules, verify that rule triggers and actions are functioning properly, and even simulate sensory input into the system.

1.2.2 Task 2: Finalize Sensor-Motion System and Related Systems

1.2.2.1 Vision System

Objective: The specific deliverable is the ability to reliably recognize all of the instruments in the minor instrument tray.

This objective was completed successfully. We have developed and thoroughly tested a general purpose vision system capable of identifying, locating, and determining the orientation of objects laying within the camera's field of view. This task involved the design, development, and fabrication of the hardware components of the system – the digital cameras and the camera post –

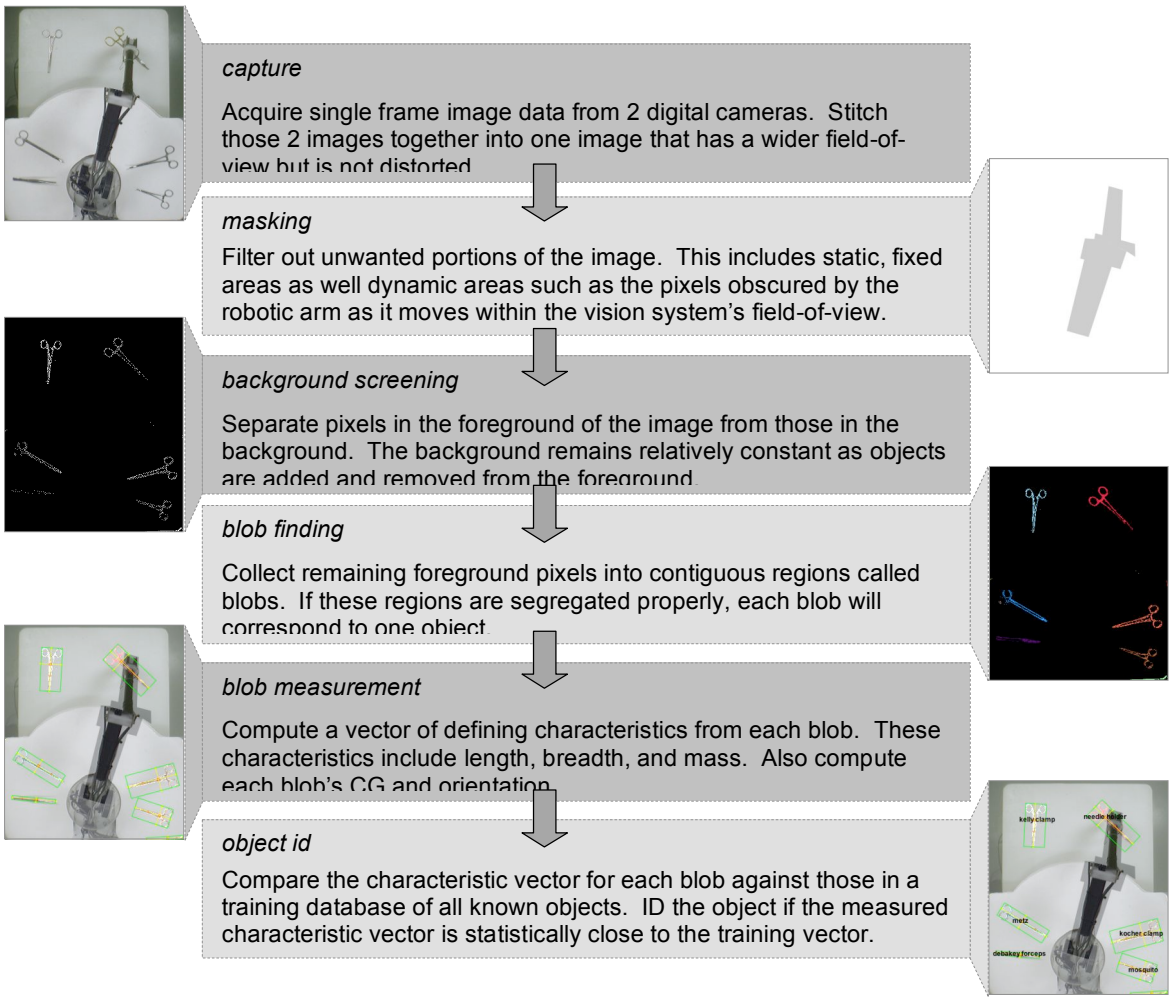

Figure 5. Vision System Layers

and the design and implementation of the software components. The software components of our vision system are shown in Figure 5.

Our vision is based on two assumptions: that all instrument types to be identified are known in advance and that the background of the images is relatively static. Our system requires training on the visual signature of each instrument type. Note that this training is required only for each instrument *type* and not for each instance of that type. Note also that this training is only required once and can be performed when the robot is first fielded. The requirement that the background be largely static is achieved by our hardware architecture. The digital cameras point directly down onto a set of instrument trays also attached to the robot. This whole unit straddles the OR table above the patient. The field-of-view of the cameras is limited to this area which essentially the robot's operational envelop. This means the robot is generally the only thing moving in this area. Thus the background changes little.

1.2.2.2 Motion System

> *Objective: The specific deliverable is the ability to move an instrument from the various instrument zones to the surgeon's hand in times comparable to those of a human scrub technician. The times used for this comparison will be those obtained from the testing procedures outlined in Stage 1 of Task 3.*

This objective has been partially met at this time. We are continuing to increase the speed of the robotic arm, but have not yet been able to consistently match the speed of a human scrub technician. Section 3 below describes this in more detail.

Delivery times for a human scrub technician vary. The experienced scrub that has everything properly set up and is not involved in other functions can deliver an instrument in 1-2 seconds. However, a less experienced scrub, or one not familiar with the procedure or surgeon's preferences, or one involved in counting instruments or other activities, may take significantly longer. If we assume that there are about 16 instrument deliveries in a typical minor case (as was the number in our first clinical trial with the robot), the total time for the robot to deliver these instruments would be 198.4 seconds (16 times 12.4) or 3.31 minutes. The human scrub person (best case scenario of 1.5 seconds/delivery) would take 24 seconds (1.5 times 16) for these instrument deliveries. The robot would therefore add 2.91 minutes over the time spent by a human scrub person under ideal conditions. This is in the context of a typical total surgical time of 32 minutes (as occurred for our first case). The extra 2.91 minutes fall within the range of variability from other interruptions such a shift changes and personnel breaks. In fact, we hope that such interruptions will be obviated in the future by using the robot. In addition, the robot's time of delivery will improve as experience is gained in using it.

1.2.2.3 Voice Recognition System

> *Objective: The specific deliverable is a clinically useful voice recognition ability resulting from system improvements and microphone selection.*

This objective was completed successfully. We evaluated several microphones and several voice recognition packages as part of our Phase II effort. We settled on a headset microphone worn by the surgeon and a voice recognition package called Sphinx from Carnegie Mellon University. Sphinx is a speaker independent recognition system and is considered to be among the best

available. We are using the latest release of Sphinx which is a Java-based package. We have integrated it into our Java codebase and are now running both applications inside one Java virtual machine.

1.2.2.4 Other Sensors

> *Objective: The specific deliverable is a sensor to monitor the presence of an instrument on the gripper.*

This objective was completed successfully. We have designed and implemented a *bump* switch on the end of our electromagnetic gripper. The surgeon must engage this switch by lifting up slightly on the instrument before the system will turn off the electromagnet. We know then that the surgeon hand is supporting the instrument and it is safe to release it. While this is not exactly a "sensor to monitor the presence of an instrument," it is a viable alternative. We found this design solution to be easier to implement and more reliable.

1.2.2.5 Circulating Nurse Interface

> *Objective: The specific deliverable is a software application running on a separate computer, which handles a set of inputs/outputs between circulating nurse and robot, and is connected to the robot's computer.*

This objective was partially completed. We purchased a handheld computer, programmed it with some basic functionality the circulating nurse will require, and established a wireless connection between the device and Penelope's main computer. Due to time and staffing limitations however, we were unable to incorporate all the desired functionality on the handheld device. We were able to achieve the most difficult aspects of the objective – learning how to program the device and establishing the wireless connection. We feel then that the remaining tasks of finalizing the graphical user interface on the device will be achievable as we continue developing the product.

1.2.3 Task 3: Full System Validation

> *Objective: The specific deliverables are the stage one validation (objective criteria for instrument delivery time requirements as compared with human scrub nurses) followed by stage two validation (objective criteria for high-level performance requirements), leading to a specific final deliverable that the robot's overall performance will be clinically acceptable.*

This objective was completed successfully. We in fact exceeded our objective by completing our first clinical trial with the robot during our Phase II effort. Penelope's first clinical operation was on June 16, 2005 at the Allen Pavilion of the NewYork-Presbyterian Hospital. This historic event was the first autonomously assisted surgery. Penelope assisted in the removal of a benign tumor from the left arm of a patient. Spencer Amory, chief of surgery at the Allen Pavilion, performed the procedure and gave Penelope voice commands for instruments. The procedure lasted approximately 32 minutes and Penelope completed 37 instrument maneuvers. The robot performed all of its assigned functions properly. "Penelope worked smoothly and efficiently to provide the correct instruments for the procedure as I asked for them," said Dr. Spencer E. Amory. "Although the procedure was relatively simple, I believe this marks a big step in the advancement of robotics and the introduction of advanced technology to the operating room."

Prior to this clinical validation, we conducted integrated testing using the actual robot – with all rules enabled – according to predefined test plans with expected results. As we provided input to the robot, we monitored its behavior and evaluated its performance. In addition we generated detailed rule trace output during validation testing which was analyzed later to verify that expected rules were fired with the expected results. The rule trace is a timestamped list of every rule firing with a human readable description of the rule's action. An example rule trace is shown in Figure 6 below.

```
  740: Actualize a goal to deliver the Richardson Retractor.
   77: Step: Speak phrase "Richardson Retractor".
   56: Step: Move arm from (0.275 0.0 -0.07) to (0.0 0.25 -0.075).
  137: Step: Finish waiting for properties (:motionTargetReached true) to appear.
  177: Step: Move arm from (0.275 0.0 -0.07) to (0.0 0.25 -0.115).
  173: Step: Finish waiting for properties (:motionTargetReached true) to appear.
   98: Step: Set the magnet power to 0.8 (object on magnet: Richardson Retractor).
   99: Step: Call function (assertInto intrumentCacheState :cache :gripper).
   35: Step: Move arm from (0.275 0.0 -0.07) to (0.0 0.25 -0.055).
  187: Step: Finish waiting for properties (:motionTargetReached true) to appear.
   32: Step: Set the magnet power to 0.8 (object on magnet: Richardson Retractor).
   31: Step: Move arm from (0.275 0.0 -0.07) to (0.5 -0.25 -0.04).
  144: Step: Finish waiting for properties (:motionTargetReached true) to appear.
   80: Step: Call function (setState :waitingForBumpSwitch true).
  127: Running expiration timer for :waitFor to expire in 4 seconds.
 2538: Cache new arm position, (0.5 -0.25 -0.04), in system state.
 1554: Retracting expired :waitFor.
   12: Step: Skip :gotoLabel step due to failed execution condition.
   28: Step: Express goal :instrument.discard.
   29: Sorting goals according to priority.
   57: Actualize a goal to discard an instrument
   58: Step: Skip :gotoLabel step due to failed execution condition.
   48: Step: Move arm from (0.5 -0.25 -0.04) to (0.4288 0.1924 -0.12).
   84: Step: Finish waiting for properties (:motionTargetReached true) to appear.
  112: Step: Move arm from (0.5 -0.25 -0.04) to (0.4288 0.1924 -0.17).
   85: Step: Finish waiting for properties (:motionTargetReached true) to appear.
   78: Step: Set the magnet power to 0.0 (object on magnet: nothing).
  114: Step: Call function (assertInto intrumentCacheState :cache :transferZone).
   25: Step: Move arm from (0.5 -0.25 -0.04) to (0.4288 0.1924 -0.08).
   47: Step: Finish waiting for properties (:motionTargetReached true) to appear.
   21: Step: Ignore label :magnetIsEmpty.
  119: Step: Move arm from (0.5 -0.25 -0.04) to (0.275 0.0 -0.07).
  105: Step: Finish waiting for properties (:motionTargetReached true) to appear.
   11: Step: Acknowledge that goal :instrument.discard has been completely fulfilled.
   10: Step: Goto label :G547041312180974.
   94: Step: Ignore label :G547041312180974.
   42: Step: Call function (assert goals :goal :findInstruments :priority :system).
   18: Sorting goals according to priority.
  127: Step: Acknowledge that goal :instrument.delivery has been completely fulfilled.
```

Figure 6. Instrument Delivery Rule Trace

The rule trace output shown in Figure 6 gives a good indication of the level of detail available to us for validation. Each line corresponds to one rule firing. The first line establishes the goal of delivering a Richardon Retractor to the surgeon. The subsequent list of "Step" rule firings are essentially a program that *actualizes* that goal. They are an ordered list of things to do in order to

accomplish the goal. Some steps must complete before the step program progresses, while others launch off threads that fulfill the step's actions in parallel with the rest of the step program. The second line "parrots" the name of the instrument back to the surgeon as a cross check. Note that the next step executes just 56 milliseconds after that "Speak phrase" step. This is an example of the latter type of step that needn't finish before the step program progresses. Thus we can begin moving the arm around to pick up the instrument before we're finished announcing its name. All support for these step programs is implemented as part of our rule set and CA. We can continue to analyze these rule trace lines one-by-one to validate that the expected action was taken at each step.

2. Problems Encountered And Resolved

We revised our full system validation strategy during the course of our Phase II effort. Prior to Phase II we didn't believe we would be able to validate the system in an actual surgery. In our proposal we described means of validation using an inanimate abdominal surrogate in a simulated surgical environment. As we were experimenting with this technique we learned that it could be possible to move more quickly than expected to clinical testing in a real OR. We then refocused our energy on that validation method. We applied for and received an Investigational Device Exemption (IDE) from the Columbia University Institutional Review Board (IRB). This allows us to use the robot in the OR while the FDA 510(k) application is pending. As part of this review, we generated a large amount of detailed documentation describing the safety of both our robot and test protocol we used to validate it. This shift in strategy had two benefits. First, it allowed us to validate our device in a more realistic fashion. Second, the documentation it required, and the IDE itself, will form the basis for our FDA application. We have recently been granted an extension on our IDE which will allow us to continue our clinical testing in 2007.

3. Unresolved Problems

We have been unable to increase the speed of our robotic arm as much as we had expected. This is an important factor, as it effects our instrument delivery time and thus surgeon acceptance. We encountered two problems in this area, both related to the design of this first version of our robot. First, our strategy for commanding motion in the stepper motors that move the arm appears to be flawed. This subsystem is comprised of Java code, C++ code, firmware, and electronics. Problems with the technique we use in the firmware and electronics have made it difficult for us to increase the speed of the arm without sacrificing accuracy. The second problem involves our electromagnetic gripper. This type of gripper has many advantages based on its simplicity. It was easy to design and fabricate and it is reliable and inexpensive. The problem however is that a magnet's gripping strength is far weaker in the shear direction (perpendicular to the magnetic field). This can result in instruments spinning slightly if the robot's arm is accelerated too rapidly.

We are addressing both of these problems in the next version of our robot. We are redesigning our motion control electronics to fix the first problem. We have identified the source as a failure mode in which the stepper motors can continue to move beyond the intended period of time if the higher level software is slowed down by unrelated, but parallel, processing requirements. This unintentional motion is small, but it causes positional inaccuracy in the arm. We will use a revised strategy in the next version which will prohibit this possibility. By resigning our firmware and electronics, we can ensure that the motors cannot move beyond the intended period of time, regardless of the load on the processor.

The second issue will require a redesign of our gripper. We have identified three candidate designs to address the issue: one that uses suction, one a pneumatic actuator, and one design that retains the electromagnet but uses a different shape to make the gripping surface more compliant. We have secured additional funding from the US Army's Telemedicine and Advanced Technologies Research Center address this and other issues.

4. Serendipitous Results

Several unanticipated business opportunities have emerged as a result of our Phase II efforts. The first was an important role in the Trauma Pod project sponsored by DARPA. This program is exploring the possibility of an unmanned surgical facility for use in a frontline military setting. RST was selected to contribute the machine vision system for identifying and counting surgical supplies. This has proved to be an excellent chance for us to leverage our Phase II development effort to expand our business opportunities.

We are also exploring unexpected benefits of the cognitive architecture we designed in Phase II. As a design goal, we strived to keep the CA and Penelope's Java codebase as general as possible. As described above, all detail specific to the task of serving surgical instruments – and indeed all detail specific to the Penelope robot itself – has been abstracted from our core system. This detail is entirely encoded in the robot's rule set. As a result, we have realized that we have developed not only a robotic scrub technician, but also a general-purpose robotics framework which could form the basis of an entirely different system. We are currently retooling our business plan to include this framework and our integration expertise as a separate product line.