Hierarchical Ada Robot Programming System (HARPS):

A Complete and Working Telerobot Control System Based on the NASREM Model

Stephen Leake, Lead Engineer; *National Institute of Standards and Technology (NIST)*
Tom Green; *Digital Equipment Corporation*
Sue Cofer, Lead Author; *Digital Equipment Corporation*
Tim Sauerwein; *NASA Goddard Space Flight Center*

Abstract

HARPS is a telerobot control system which can perform some simple but useful tasks; this capability is demonstrated by performing an ORU exchange demo. HARPS is based on the NIST (formerly National Bureau of Standards, NBS) NASREM model. The primary programming language for all developed software is Ada, and the project incorporated a number of different CASE development tools. HARPS, implemented at the NASA Goddard Robotics lab, integrates several on going efforts at the Goddard Robotics lab.

The results from this effort are not surprising. NASREM was found to be a valid and useful model for building a telerobot control system: its hierarchical and distributed structure creates a very natural and logical flow for developing and implementing large, complex, and robust control systems. Similarly, the ability of Ada to create and enforce abstraction was found to enhance the implementation of such a control system. The CASE tools utilized showed some promise in helping to design a large system which processed a tremendous amount of data involving very complex computations and relationships.

An overview of NASREM, the ORU exchange demo, the HARPS system, and the development tools used in HARPS is given in this paper.

1 OVERVIEW OF NASREM

The NASA Standard Reference Model (NASREM) was developed by the National Institute of Standards and Technology (NIST, formerly National Bureau of Standards (NBS)) to serve as a model for implementations of telerobot control systems for the Space Station: it is intended to be used as a reference document for the functional specification of the Initial Operational Capabilities (IOC) Flight Telerobot Servicer (FTS). NASREM defines a logical computing architecture for telerobotics in general. At the time it was developed, NASREM incorporated research and development work sponsored by several groups, including JPL, Langley Research Center, JSFC, MSFC, Ames, DARPA, Wright Patterson Air Force Base, MIT, and NIST. NASREM incorporates many of the concepts explored in these projects, such as goal decomposition, hierarchical planning, model driven image analysis, blackboard systems, expert systems, multivariant state space control, reference model adaptive control, dynamic optimization, and learning systems.

A diagram of the functional system architecture for NASREM is shown in Figure 1. NASREM is hierarchically partitioned into six control levels; a different fundamental mathematical transformation is performed at each level. In level 6, satellite servicing mission plans are decomposed into high level service bay actions which operate on groups or batches of parts. In level 5, these general actions are decomposed into lower level tasks, typically operating on individual objects. Level 4 is where each task is decomposed into specific elementary level motions. In level 3, objects are identified, and elementary moves are decomposed into strings of intermediate poses: a pose specifies both position and orientation. In level 2, dynamics are computed, and in level 1, coordinates are transformed and outputs are servoed.

The NASREM architecture is also partitioned functionally into three groups: task decomposition, sensor processing, and world modeling. Sensor processing (G) modules detect events and recognize objects, or patterns, by filtering and integrating sensor data; this is often called sensor fusion. In NASREM, some of the sensor processing G modules compute confidence factors and probabilities of the correctness of the sensed data. Task decomposition (H) modules take the high level goal and decompose them into low level actions, which it will then execute. Each task decomposition (H) module has a job assignment manager, a set of planners, and a set of executors. The world model (M) modules maintain the common memory knowledge base, and potentially provide predictions of expected sensor data, provide current state data to task decomposition execution modules, and build a hypothetical state for the task decomposition planning modules.

The operator interface allows a human operator to observe and supervise the telerobot. This can be through various devices, such as joysticks, mouse, keyboard, voice I/O, and CRTs. NASREM allows the operator to interface at any level in the hierarchy, with any functional module, at any time. The operator may choose to perform a task under complete teleoperation control, have the robot operate under automated control, or a combination of both control modes, called mixed mode. [1] [2]

2 HARPS PROJECT GOALS

The primary goal of the HARPS project was to build a complete and working telerobot control system based on the NASREM model in order to prove the correctness and usefulness of the concepts presented in NASREM: HARPS is, in fact, the first working NASREM prototype developed with NIST involvement. To demonstrate completeness the system does all aspects of a small but useful demo using vision and force sensors to assist in the transfer of an object from an unknown position to a second position, switching between autonomous and teleoperation.

Ada was chosen as the primary programming language; this was, in part, mandated by existing NASA policies that Ada be used as the language for all systems running on the Space Station. HARPS, therefore, became a vehicle for evaluating not only the usefulness of Ada programming structures but also the ability of Ada to be used as a telerobot control language: is it fast enough, and do the software engineering concepts inherent in the language lend themselves to the development of very large and complex real-time systems, such as a complete, sensor integrated, telerobot control system. In addition, a number of different Computer Aided Software Engineering (CASE) tools were

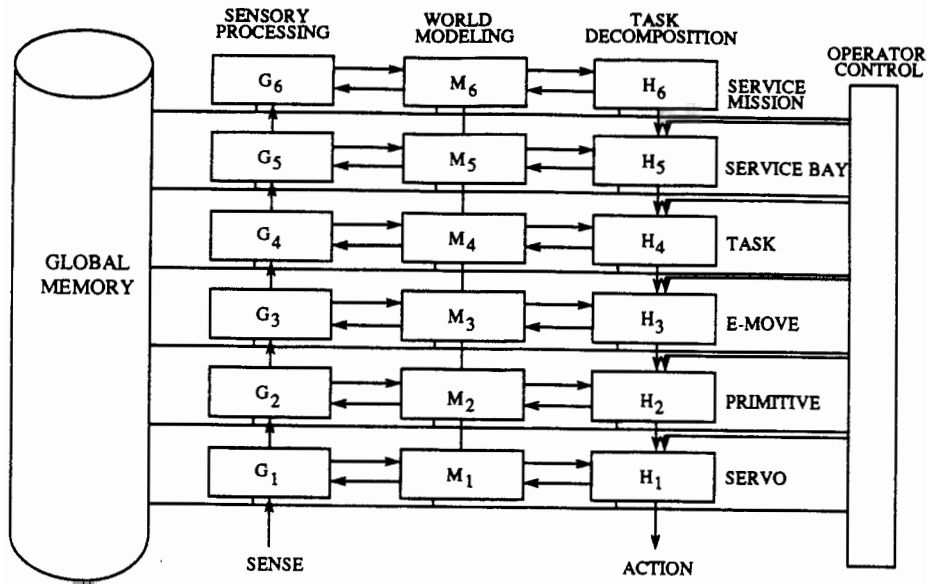

Figure 1
NASREM: A Hierarchical Control System Architecture

evaluated during the project to test their usefulness in the HARPS design process.

Another objective of HARPS was to incorporate existing hardware and software into the system, trying not to reinvent the wheel as much as possible. HARPS is highly modularized; this allows easy insertion of subsystems into HARPS as they are developed, or modified to fit in the case of existing hardware/software. A detailed description of which modules have been developed by the HARPS engineering team, and which were incorporated from existing systems, is given in the following sections.

Finally, HARPS unifies several efforts at the Goddard Robotics lab, serving to integrate the results of those efforts, and providing a solid base for future development for the Flight Telerobotic Servicer at Goddard.

3 HARPS SCOPE

HARPS is an implementation of the first 3 control levels described in NASREM: SERVO, PRIM, and E_MOVE. The task decomposition, world modeling, and sensor processing functionality as described in NASREM are included, as well as a workstation to serve as the operator interface described in the model.

4 DEVELOPMENT APPROACH

A major challenge in the design of HARPS was to take the standard NASREM architecture, and give it meaning in the context of our specific system under development. We approached this problem from a number of different angles. First we viewed the project as a standard development of a real- time S/W control system. Data flow design methodology (Ward & Mellor real- time extensions) was applied using a standard CASE tool so that the system boundaries, external events, and leveled data and control flows were defined. Once this data flow model was developed, a number of informal design walk-thru's and reviews were held to both shape the design to the NASREM mold, and to establish compliance to it.

Secondly, in addition to the data-flow model, module interface descriptions were outlined as part of a traditional design document [3]. This design document led naturally to a formal Ada design specification outlining package layout and tasking structure.

A very important step in developing the final NASREM-compliant design was to establish a reasonable Task Decomposition schema (see Section 7.3.3). This was accomplished manually by simply dividing the observable top-level steps of the existing RCCL script in a logical way into E-MOVE and PRIM command decompositions using appropriate NASREM classification guidelines. Although key to the design, this derivation was based to a large extent on common sense, and therefore was somewhat arbitrary. Further clarification of the NASREM task decomposition guidelines would enhance future automatization of this deriviation process.

5 OVERVIEW OF ORU EXCHANGE DEMO

A simple demo, the ORU Exchange Demo, was programmed in HARPS. The demo illustrates the validity of the concepts outlined in the project goals while providing a base for future work. The Flight Telerobotic Servicer (FTS) is emulated by a PUMA 762 robot, which is equipped with a wrist-mounted camera, a wrist force/torque sensor, and an end-effector capable of grasping the ORU handles. There is a sensor in the end-effector showing its latch state: after mating with the ORU, the end-effector must latch before it can pick up and move the ORU. The ORUs are approximately 3' x 3' x 3', with two ORU handles on top: one to hold the ORU while transporting it; the other to open the ORU door to expose the electronics inside. Also on top of the ORU is a machine vision target, consisting of four black blobs on a white background; the target is used to calculate the pose of the ORU with respect to the arm, thereby locating the ORU handles. The platforms have 3 features (holes) that compliment 3 features (pegs) on the ORU. These features are used to insure that the ORU is correctly placed on the platform: there are binary sensors in the 3 platform features showing when the ORU is correctly docked, or mated with the platform. There are 3 cameras set up around the platform to allow the operator to monitor the robot's operations from several different perspectives. The operator interface consists of an ANSI terminal, a graphics terminal, two non-force-reflecting joysticks for teleoperation of the robot, a video display

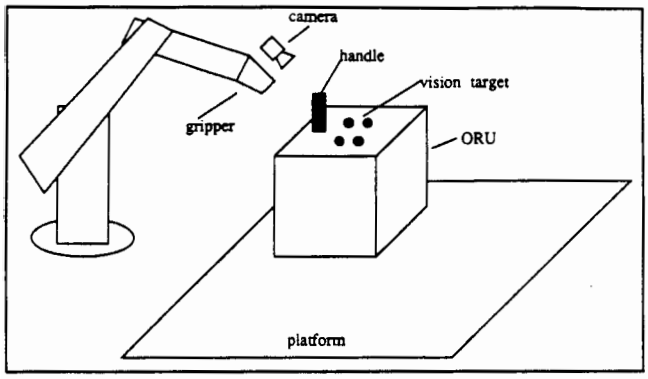

Figure 2
ORU Exchange Demo

screen for the wrist camera, and video display screens and joysticks for the monitoring cameras around the platform. Only the wrist camera is available for machine vision.

To begin the demo an ORU is sitting on a source platform, ready to be moved to a destination platform; the position of the ORU relative to the robot is unknown. The robot is in a pre-defined park position. The operator tells the system to transfer the ORU to the destination platform. The system requests help from the operator in locating the ORU at the source platform. The operator uses the joysticks to position the robot arm so that its wrist camera can see the ORU target. The operator then uses the joysticks to draw a bounding box on the machine vision monitor around the four target blobs, and gives control back to the system. The vision system uses the bounding box to reduce unnecessary background noise and focus on the four target blobs; the system then servos to the target and, having locked in on the target, the pose of the ORU and its handles are calculated and stored in the world model. The system generates the commands for the robot to mate with the ORU movement handle, extract the ORU from the source platform, and move it near the destination. The system then requests help from the operator in mating the ORU to the destination platform. The operator uses the joysticks to mate the ORU to the platform, and the system withdraws the end-effector. This completes the transfer. The operator then tells the system to open and close the ORU door, which would allow required servicing within the ORU; finally, the operator tells the system to release the ORU door handle and return to it's park position, completing the ORU exchange demo.

The operator plays two roles, both as a supervisor providing task-level decisions and commands, and as a resource supplying vision processing and manipulation skills; thus the operator completes the skill-set required to do the ORU exchange by supplying those skills currently not automated in HARPS. It would certainly be feasible to have two separate people acting in the operator capacity: a supervisor to issue commands, and an operator to provide vision and manipulation skills.

6 HARPS SYSTEM OVERVIEW

The HARPS control system design is based on NASREM [3] ; it includes those NASREM modules needed to implement this demo, and incorporates as much currently available hardware and algorithms as possible.

The HARPS modules are functionally partitioned as in NASREM into three main sections: task decomposition, sensor processing, and world model. Task decomposition modules compute commands for actuators, sensor processing modules compute measurements of the world, and the world model stores these measurements in a useful format and provides the mechanisms (servers) for the other modules to access this information. The functional group into which a module belongs is usually obvious: for example, the velocity of the manipulator as it moves toward an object is computed in a task decomposition module because it is a command to the manipulator. The pose (position and orientation) of the object toward which the manipulator is moving is computed in a sensor processing module if it is measured using the camera mounted on the manipulator. The object pose, once calculated, and the mechanisms for accessing it, are stored in a world model module. Sometimes, however, it is not immediately clear to which functional group a module should be assigned. For example, the module which controls a camera focusing on a object may appear to be a sensor processing module, since it is concerned with focusing a sensor; however, focusing is accomplished by sending an actuator command to the focus servo and therefore belongs in task decomposition.

The task decomposition modules are hierarchically separated into the NASREM control levels: SERVO, PRIM (primitives), and E_MOVE (elemental moves). For every actuator, there is a task decomposition SERVO module which monitors the action of that actuator and uses sensor information (retrieved from the world model) to verify that commanded actions were taken: typically, servo algorithms which came with actuators fit the HARPS implementation of NASREM and are used unchanged. These SERVO modules take commands with simple value parameters (numbers), while E_MOVE and PRIM modules take commands with object parameters.

The operator interface as described in NASREM has been divided into two separate functions: supervisor and operator. The supervisor issues high level task commands to task decomposition. The operator provides the low level skills required in performing tasks, such as the ORU exchange demo described above.

All access to the world model data goes thru server modules which guarantee correct access, resolve read/write conflicts, and transform data to the desired format. Sensor processing servers update information in the world model resulting from sensor measurements; in the future, these servers will also provide sensor fusion. Task decomposition servers retrieve data, transform the data into the required reference frames, and extrapolate to the requested time.

Most planning is done off-line, and the resultant plans are stored in the world model. NASREM does not provide for the creation or execution of sensor plans, so HARPS extends the concepts related to task planning to handle sensor planning. Plans for doing tasks, and for using sensors, are created off-line and stored in the world model. Task Decomposition plans are then executed directly by Task Decomposition modules. Sensor plans are executed indirectly by the Task Decomposition modules, by sending messages to appropriate Sensor Processing modules. Thus all control in the system remains in the Task Decomposition modules. As HARPS is expanded, the planning can be done in automated routines and then stored in the world model for later use; the existing routines to extract and execute these plans would be used, then, with little or no alteration.

7 DETAILED DESCRIPTION OF HARPS MODULES

A detailed description of several modules implemented in HARPS follows. A bottoms up approach is taken, where the lower levels are described first. A detailed description of all modules and module functions is outside the scope of this paper; more detail is available in [3].

7.1 Sensor Processing

There are three levels in Sensor Processing as shown in Figure 3. The data acquisition level is the lowest level, and all sensors used in HARPS have a data acquisition module. Only the camera has modules for the two higher levels: low, and intermediate. The vision processing system consists of a frame grabber installed in a microVAX. There are three vision processing modules: FILTER, SEGMENT, FEATURES. The vision system is coded in PASCAL, and interfaces to the rest of HARPS via DECNET. It can process an image in approximately one second. This system was originally implemented in RSL [5], and was ported to the microVAX and PASCAL as part of the Goddard RCCL ORU exchange demo.

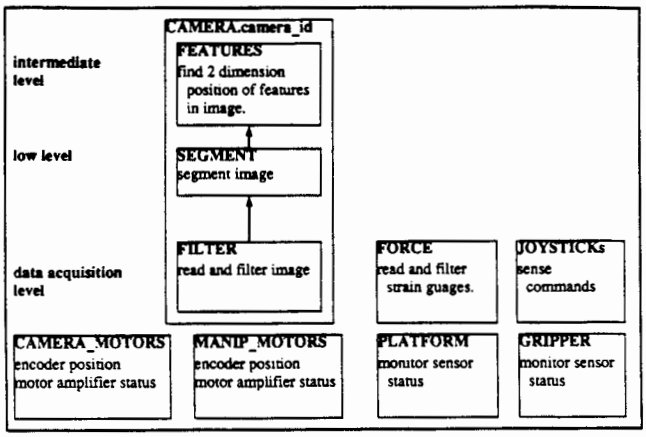

Figure 3
Sensor Processing Modules

7.1.1 Data Aquisition Level

There are seven data acquisition modules. FORCE reads the strain gauges on the wrist sensor, and does some filtering. MANIP_MOTORS reads the joint encoders and motor amplifier status. The motor amplifiers report binary current limit information. CAMERA_MOTORS reads the camera motor encoders. ORU_GRIPPER returns the gripper status, which indicates that the gripper is either latched or unlatched. PLATFORM reads the platform limit switches, which tell whether an ORU is docked. The two joysticks used in teleoperation share the JOYSTICK modules used in HARPS: camera, manipulator and bounding box. Each module reads the corresponding joystick and any joystick buttons, and sends the information to the World Model. FILTER reads a frame from the vision system and filters it.

7.1.2 Low level

The SEGMENT module iconically segments the image frame from FILTER. In the HARPS implementation, segmentation is accomplished by thresholding.

7.1.3 Intermediate level

The FEATURES module finds the 2D position of the features in the segmented image of the target. The target used in the HARPS ORU exchange demo is four black circles on a white background; the black circles, or blobs, are the features to be located by FEATURES. FEATURES chooses between two different algorithms, coarse acquisition and fine acquisition, based on whether the target position is well known or not. Coarse acquisition is used when the target is not well known, and can therefore incorporate very little existing world model data. Coarse acquisition uses a standard connected components algorithm to locate the four blobs within a bounding box: if no bounding box is present, one is requested from the operator.

Fine acquisition is faster and more accurate than coarse acquisition. It is used when the target position is well known, which means that the actual location of the four blobs will be within a certain neighborhood of the predicted location of these features. The fine acquisition algorithm gets from the world model the predicted location of the features in the image, puts a window around each predicted location (which assumes that the actual image feature is within this window), and finds the first brightness moment within each window, thus locating the actual features. The size of the window around each feature varies, depending on the required speed and accuracy.

7.2 World Model

Figure 4 shows the internal structure of the world model, along with the servers (called processes in the figure) that provide access to the world model for Task Decomposition and Sensor Processing modules. This structure is internal to the common memory described in NASREM.

Each storage module stores geometry and mass information on the object it models, together with any algorithms appropriate to the object, such as kinematics. The World Model, then, is a *knowledge base*, not just a data base. Each server is an independent package, one for each module that needs to access the World Model. The storage modules are not processes themselves: if they are packages which contain code, then this code is executed by a server process. This code can be executed in parallel by many different servers.

The World Model stores 6D information whenever it is defined: 6D includes 3D for position and 3D for orientation, and is often called "pose" in this paper. Only objects capable of independent motion store dynamic terms, such as relative position and velocity. All requests for time-dependent information will have a time stamp from a global clock, providing time synchronization. This concept is presented in NASREM and is newly developed in HARPS.

7.2.1 Servers

There is one server for each Task Decomposition and Sensor Processing module that accesses the World Model: the basic function of the server is to manipulate storage. Server modules are named for the task decomposition or sensor processing module that they serve. If the E_MOVE level FTS module in Task Decomposition requires data from the World Model it may issue one of many requests to the TD.E_MOVE.FTS server. GET_PLAN(plan_type, qualifiers), for example, returns the plan for mating, demating, or opening; the qualifiers specify object and environment data, depending on the plan. GET_POSE(object) returns the pose and associated error bounds for the object specified. FIND(object,tolerance) requests that the FEATURES vision processing module find the requested object, and update its pose in the world model.

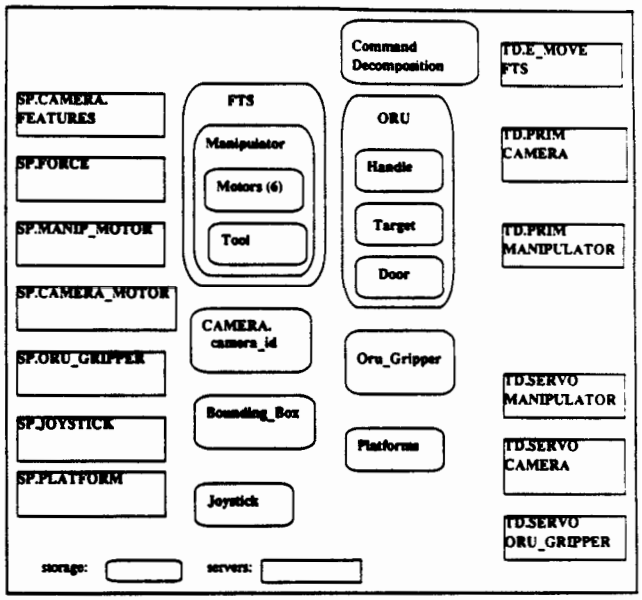

Figure 4
World Model Modules

If the PRIM level MANIPULATOR module in Task Decomposition requires data from the World Model it may issue one of several requests to the TD.PRIM.MANIPULATOR server. The UPDATE_POSE(object) request is a subset of the FIND(object) algorithm; however, UPDATE_POSE runs continually until an END_UPDATE_POSE(object) is issued. The GET_FORCE(object) request returns the forces on the object. GET_COMPLIANCE(object) returns the effective compliance, which is computed by summing the compliances of all the objects in the kinematic chain that includes the given object. GET_TRAJ_PARAMS(object) returns velocity and acceleration limits, as required by the Joystick module. GET_VEL(object) gets the velocity and associated error bounds of the object. INVERSE_KINEMATICS(cartesian_point,configuration) performs the manipulator's inverse kinematics algorithm to convert the cartesian point into motor or joint space. The configuration is used to resolve ambiguities in kinematics, such as elbow up or down.

The TD.PRIM.CAMERA server module passes information to the PRIM level CAMERA module in Task Decomposition. One request is GET_VIEWED_SIZE(object), which determines the apparent size of the object or feature in the camera image; this is used to determine the appropriate zoom setting.

The TD.SERVO.MANIPULATOR server for the SERVO level MANIPULATOR module in Task Decomposition supports only one request, GET_POSE_VEL(motor), which returns the current encoder count (position) and velocity of that manipulator motor.

The SP.CAMERA server, which interfaces the Sensor Processing camera modules to the world model, supports requests like UPDATE_POSE(object, features), which updates the pose of an object, by matching the features to the object model. For the case of the 4 blob target, the model matching algorithm is taken from [4] .

7.2.2 Storage Modules

Storage modules are Ada packages or data structures containing the data necessary for Task Decomposition and Sensor Processing modules to do their tasks. Algorithms for determining dynamic information concerning an object may be present with the storage module of that object, but these algorithms would be executed by the requesting server. The storage modules and servers are shown in Figure 4.

7.3 Task decomposition

Figure 5 shows the Task Decomposition hierarchy. The vertical levels correspond to the NASREM Task Decomposition E_MOVE, PRIM, and SERVO levels, while the horizontal divisions reflect individual hardware components: FTS includes the manipulator and camera motors, and the gripper actuators. There are task decomposition modules described in NASREM which are not present in the current HARPS implementation. The gripper PRIM level module has no specific functions in HARPS and is therefore not included. There is no Camera E_MOVE module because it is not capable of complex tasks. The supervisor performs the functions of the TASK and higher levels in NASREM.

SERVO modules take commands with simple value parameters (numbers), while E_MOVE and PRIM modules take commands with object parameters. For example, the PRIM CAMERA module takes the command *FOCUS_ON (object)*, while the SERVO CAMERA module takes the command *SERVO_FOCUS(encoder_tics)*. Similarly, the PRIM MANIPULATOR module takes the command *AIM_MOVE (sensor, target, goal)*: sensor and target are object names. The SERVO MANIPULATOR module accepts the command *SERVO_MOTOR (encoder_tics)*.

7.3.1 SERVO level

Manipulator

There are six joints, and therefore six independent manipulator SERVO level modules: these are the joint servo algorithms which came with the robot. They consist of simple PID joint position servo algorithms, sending currents to the motors. In addition to the servo algorithm, each module attempts to control the motor dynamics in a simple way. Each step input is linearly interpolated at the servo rate. The motor current is limited, and if the motor gets outside a preset following error, it is shut off and the motor breaks applied. The only input command to the manipulator SERVO level is SET_POINT(desired_position), which issues a simple PID joint position servo algorithm, sending current commands to the motors.

ORU gripper

There are two independent gripper modules, one for the latch actuator and one for the screw actuator. The two latch commands are LATCH and UNLATCH, and the two screw commands are TIGHTEN_SCREW(turns) and UNTIGHTEN_SCREW(turns).

Camera

There are three independent SERVO level camera modules for each camera: iris, focus, zoom. They consist of the PID position servo algorithms which came with the hardware, sending current commands to the camera iris, focus, and zoom motors. The only input command to the camera SERVO level is SET_POINT(desired_iris, desired_focus, or desired_zoom).

7.3.2 PRIM level

Manipulator

A primary function of this module is to control the dynamic behavior of the manipulator. This is done by limiting the acceleration and velocity of the individual joints, as well as limiting the acceleration and velocity of the tool point at the end of the arm. The limits are currently hand-coded; when the E_MOVE and PRIM level planners are implemented, the limits will be set dynamically by the planners.

Another function of this module is low-level collision avoidance and reflex reactions. Collision avoidance at this level refers to not exceeding joint limits, and avoiding the ORU, as long as it can be located by the vision system. Reflex reactions will cause the robot to back off from any unexpected force sensed by the wrist force/torque sensor.

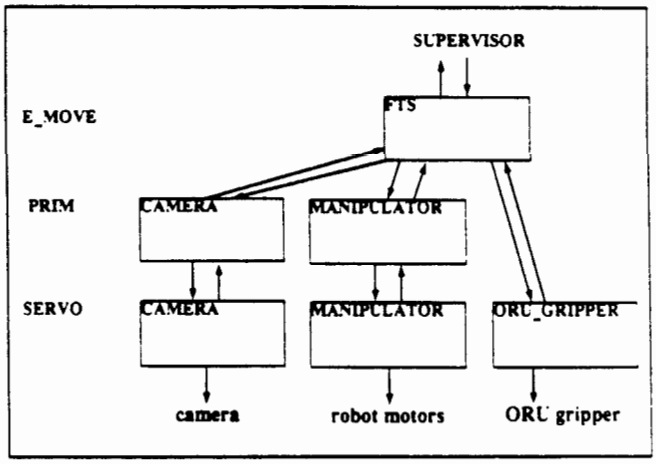

Figure 5
Task Decomposition Modules

Higher level functions performed by this module include simple path following, cartesian sensor servo for finding the ORU, force control for mating the ORU to the end effector and the platform, and non-force-reflecting teleoperation (both cartesian and joint space).

Algorithms for all these functions are taken from the existing RCCL Goddard ORU exchange demo code and the RSL system [5] There are several PRIM level manipulator input commands implemented in HARPS. TELEOPERATE reads a desired pose from the world model and drives the manipulator to this pose; the desired pose has been generated by the manipulator joystick. This illustrates a key NASREM concept: the world model facilitates the separation of two independent processes, sensor processing and task decomposition. In this case, the processing of the joystick (sensor processing) happens independently of the manipulator's motion that TELEOPERATE generates (task decomposition). Therefore, any joystick can be used to drive the manipulator. Similarly AIM_MOVE(sensor,target,goal,traj_params) uses a goal position generated simultaneously in sensor processing; this allows AIM_MOVE to maintain the sensor's view of the target while moving the manipulator which holds the sensor to this goal position.

PATH(segment_list) follows a piece-wise smooth list of segments, given in either cartesian or joint space. The path should be free of obstacles; this should be checked by the E_MOVE level, and is part of the next planned release of HARPS. PEG_IN_HOLE(held_object,environment) inserts the held object (the peg) into the environment (the hole); a simple active compliance algorithm is used. Similarly, PEG_FROM_HOLE(held_object,environment,goal) extracts the held object from the environment using active compliance.

Camera

The function of this module is to decide how to set the camera iris, focus, and zoom parameters. FOCUS_ON(object) is a camera PRIM level input command which determines the focus setting from the object's apparent size and range, as obtained from the world model. TELEOPERATE allows the operator the set the camera's iris, focus and zoom, using the camera joystick.

7.3.3 E_MOVE level

There is only one E_MOVE level module, FTS, which is based on the RCA scripts in the existing Goddard ORU exchange demo. FTS decomposes commands from the supervisor into commands for PRIM level manipulator, commands for SERVO level gripper modules, or sub-tasks. Sub-tasks are further decomposed, just as supervisor commands, until all sub-tasks have been decomposed into lower level commands. The command decomposition is retrieved from the world model; the command decomposition has been manually planned off-line, instead of dynamically planned. TRANSFER(object,destination) is an FTS input command which decomposes into the following plan: find the object, mate the end effector to the object, demate the object from its current position, find the destination position, mate the object to the destination position, and finally demate the end effector from the object. OPEN(ORU.door) decomposes into the following plan: find the door handle, mate the end effector with the door handle, nullify the forces on the gripper, calculate and follow the required path to open the door, and nullify the forces on the gripper again. Similarly, CLOSE(ORU.door), which assumes OPEN(ORU.door) has completed successfully, decomposes to: calculate and follow the required path to close the door, nullify the forces on the end effector, and demate the end effector from the door handle. It should be noted there is no freespace checking, however; this is also included in the next planned release of HARPS.

7.4 Workstation: operator interface

The major design challenges for the the operator/supervisor interface under NASREM were deciding what information from the world model to display, how to give commands to the system, and how to do both of the above according to accepted human-factors engineering principles.

By centrally locating sensor and task command status in the World Model, NASREM facilitates access to the data by the display system. Design decisions as to which data to display were made according to whether the information was supervisory (command status) or operational (sensor feedback, robot pose). It proved important from the human-factors viewpoint to provide sufficient sensor data to the operator to allow correct telerobotic operation of the ORU exchange demo without overwhelming the operator with too much data. On this point the proper design and layout of the displays proved to be crucial.

8 CONCLUSIONS

The goals of the HARPS project were to build a complete and working NASREM implementation, to determine the correctness and usefulness of the NASREM architecture, to evaluate Ada as an implementation language for large real-time systems, and to incorporate existing hardware and software as much as possible. NASREM proved to be mostly correct and useful. When implementing a full, working telerobot control system such as HARPS, however, there were a few things that were more practically viewed from a different perspective than that given in NASREM. For example, it was more accurately stated to name the sensor processing hierarchy levels as data acquisition, low level, intermediate level, and high level, instead of using the respective task decomposition hierarchical breakdown of servo, primitive, elemental move, and task level. Similarly, the world model does not fit the task decomposition breakdown. Both sensor processing and the world model are still hierarchically structured; the point is that they have their own structure, not necessarily the same as the task decomposition structure. In addition, in HARPS, the world model and common memory are fused together, not separated as in NASREM.

It also became apparent that the operator cannot practically interface to every level in the system: the appropriate operator interfaces were found to be the higher task decomposition modules and in Sensor Processing low-level modules through joysticks. These two functions were explicitly separated into the supervisor and operator interfaces.

On the whole, the main concepts in NASREM proved very useful. Its hierarchical approach lends itself to complete, working robot controllers. The use of the world model to interface sensing and control is particularly beneficial, as is the concept of cyclicly executed independent processes, distributed over multiple processors.

Another major finding of the HARPS project is that Ada is fast enough to control a robot in real-time. The PRIM MANIPULATOR module was coded entirely in Ada, along with most of the world model and its servers. All of this code runs on a single microVAX, and meets the update rate required.

Ada is also extremely useful from a software engineering viewpoint. There are over 200 individual compilation units in HARPS, all in a unified Ada library. The strong typing and data abstraction capabilities of Ada were crucial in managing a system of this size. The separate compilation of specifications and bodies was very helpful during the design phases. We wrote many drafts of high-level bodies, using low-level specifications without bodies, to be sure the specifications were complete. This saved many hours of recompilation time.

CASE tools proved useful during the design phases. It helped the designers visualize the system, and the design walk-thrus ensured that everyone had a common understanding. However, a single CASE tool was not found with enough expressive power in the design language to do a full detailed design of the control system. We continue to search for a CASE tool that provides models of packaging and abstraction as powerful as those in Ada. It would also be useful to be able to automatically execute the design model, instead of requiring manual walk-thrus. This topic could easily serve as the subject of another paper.

Incorporating existing algorithms and software packages was quite straight-forward, due to the openess and modularity of the NASREM architecture. The pre-existing vision system was designed with a NASREM-style interface in mind, and it proved simple to plug it into HARPS. The algorithms inherited from RSL and the RCCL demo were re-implemented in Ada, with only minor changes, and formed the core of the PRIM MANIPULATOR module.

9 FOLLOW-ON WORK

There are several areas of HARPS that need more development, before we have a fully functional telerobot control system. Task decomposition needs to be enhanced by adding the TASK level, along with on-line path and task planning, force reflecting teleoperation, and dual-arm cooperation. Task planning would involve both active decomposition, and planning for the use of sensors.

Sensor processing can be enhanced with more sophisticated vision algorithms, to allow finding poses of objects that do not have simple vision targets. This area can benefit from the NASREM architecture by using the world model to store and integrate partial knowledge, which can in turn guide further sensor processing. In addition, the world model can be enhanced with sensor fusion algorithms (in the sensor processing servers), and more flexible plan representations, to support on-line planning. In light of the HARPS experience, adding these new capabilities to HARPS should be straight-forward.

At the completion of HARPS, the HARPS engineering team plans to work with NIST so that NASREM can be revised to reflect what was learned from an actual implementation, making NASREM a more accurate and practical standard for robot control.

References

[1] James Albus, Harry McCain, and Ronald Lumia, NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM), December 4, 1986

[2] James Albus, Harry McCain, and Ronald Lumia, NASREM—Standard Reference Model for Telerobot Control, in *Proceedings of 1987 Goddard Conference on Space Applications of Artificial Intelligence (AI) and Robotics*, May 13-14, 1987

[3] Stephen Leake, Hierarchical Ada Robot Programming System including canonical hand-coded optimized routines for demos (HARPSichord), NASREM ORU transfer demo modules, interfaces and hardware, internal report

[4] Yubin Hung, Pen-shu Yeh, David Harwood, "Passive Ranging to Known Planar Point Sets", 1985 IEEE Conference on Robotics and Automation, March 1985, St. Louis Missouri, pp 80-85.

[5] Stephen Leake, "Robot Sensor Language", 1987 Goddard Conference on Space Applications of Artificial Intelligence and Robotics, May 13-14, 1987, NASA Goddard Space Flight Center, Greenbelt, MD.