Proceedings of the IEEE International Conference on Robotics and Automation, Nagoya, Japan, May 1995

System Design and Interfaces for Intelligent Manufacturing Workcell

Gerardo Pardo-Castellote
Stanford ARL*
pardo@sun-valley.stanford.edu

Stan A. Schneider
Real-Time Innovations Inc.
stan@rti.com

Robert H. Cannon Jr.
Stanford ARL
cannon@sun-valley.stanford.edu

Abstract

This paper introduces a design technique for complex robotic systems called interfaces-first design. Interfaces-first design develops information interfaces based on the characteristics of information flow in the system, and then builds subsystem interfaces from combinations of these information interfaces.

This technique is applied to a dual-arm workcell combining a graphical user interface, an on-line motion planner, real-time vision, and an on-line simulator. The system is capable of performing object acquisition from a moving conveyor belt and carrying out simple assemblies, without the benefit of pre-planned schedules nor mechanical fixturing.

The information characteristics of this system are analyzed, and divided into three interfaces: world state, task command, and motion commands. Detailed descriptions of the resulting interfaces are provided.

The paper concludes with experimental results from the workcell. Both single-arm and dual-arm actions are discussed.

1 Introduction

Integration of a robotic workcell has traditionally been a very time-consuming process. The result is higher costs and longer lead times. As a consequence, the number of applications for which robotic automation is economical remains small.

Some of the reasons for the complexity of the integration are:

- Controlling the robotic workcell and connecting it to the rest of the manufacturing facility requires a large software development effort. The software is usually structured as many individual modules with a high degree of interconnection: interface and monitoring stations, scheduling, control, sequencers, planning subsystems etc.
- A lot of custom mechanical "glue" is required to ensure predictable, repeatable behavior of the workcell: feeders, fixtures, specialized tooling and fixed automation.
- There is a large lead time before initial system operation, and each design iteration is slow since it requires repositioning the fixtures, teaching new robot trajectories, etc. As a result, a lot of time needs to be spent on the early design phase and there is little room for iteration.

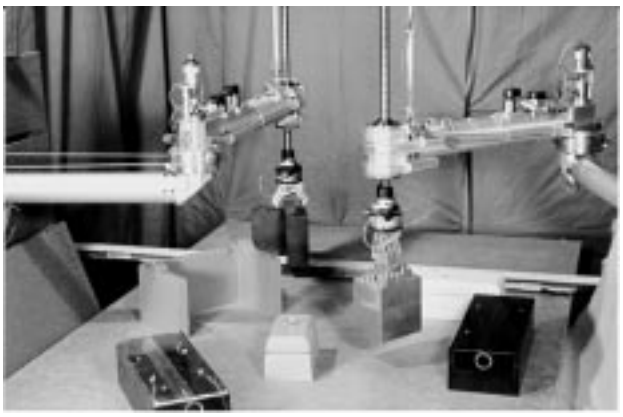

Figure 1: **The Dual-Arm Workcell**

This workcell is capable of performing vision-guided moving-object acquisitions and assemblies under the control of an on-line planner.

Mechanical glue and custom integration time can be reduced by making the cell smarter and sensor-rich so the workcell uses less structured feeding techniques (e.g. conveyors) and can plan and schedule trajectories autonomously (avoid teaching, and scheduling phases), however this exacerbates the software development and integration part.

This paper describes a novel system design and integration approach taken at the "Smart Robotic Work-

*Aerospace Robotics Laboratory

cell Project" at Stanford. It analyzes and addresses these problems and presents experimental verification of the system design in operation.

Our system relies in the use of visual tracking and on-line planning to avoid the need for fixturing or a priory scheduling. The on-line planning and control issues have been described in [6, 17]. The real-time vision system is described in [16]. This paper focuses on several new technologies developed to address the integration problem;

1. A new "interfaces-first" system-design technique (similar to the approach taken in large software projects) is emphasized over the "components-first" design technique commonly used for robotic systems.

2. Modular, parametric interfaces. The interfaces themselves are designed to be composed of multiple primitive modules which can be customized and connected in a variety of ways to create new "custom" interfaces for any subsystem modules.

3. Anonymous, stateless interfaces. The interfaces do not specify the subsystems involved, and messages are self-contained, increasing reliability and enabling replication and arbitrary connectivity.

Section 2 gives a brief description of the hardware and the experimental context that motivated the research, section 3.1 describes the new approach to system design based on the idea of starting with the interfaces (and making these modular themselves), section 3.3 describes the primitive interfaces used in the "Smart Robotic Workcell" project, and section 3.4 describes the resulting system architecture and several of the configurations that can be achieved using the primitive interfaces. The paper concludes with some experimental results on the the system's operation.

2 Overview of the Experiment

The "Smart Robotic Workcell" illustrated in figure 2 is composed of two 4-DOF arms in a SCARA configuration, a conveyor belt, an overhead vision system, and several objects that can be grasped and manipulated by the arms. Both the arms and the objects are tagged with infra-red LED's and are tracked using an overhead vision system that uses the unique 3 LED signature of each object to identify and track both the position, orientation and velocities at 60 Hz.

The experimental context of this project is the development of an architecture that allows semi-

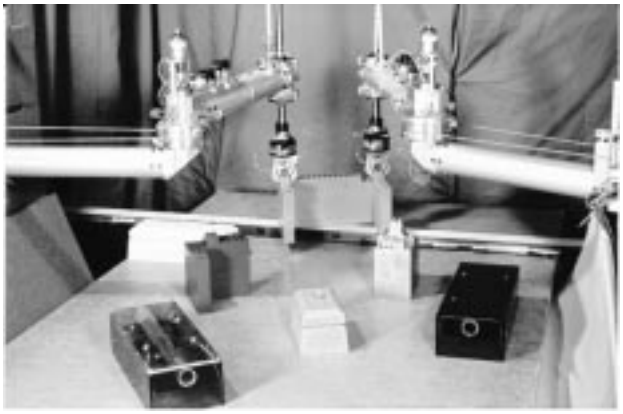

Figure 2: **Dual-Arm Motion**

The system can use both arms simultaneously, either independently or to perform cooperative motions, as pictured here.

autonomous operation of the workcell. In the experiments performed, the human specifies a high-level assembly task using a graphical interface. The assembly is specified as the relative position and orientation of the different parts with respect to the "assembly reference frame". The user also specifies the location of the assembly within the workcell.

The objects required for the assembly may either be present or brought in on the conveyor. Some of the objects can be grasped with a single arm while others (due to their weight or shape) require dual-arm manipulation. Additional objects that cannot be moved by the robot constitute obstacles that the robot must avoid.

Once the goal is specified, the user need not interact with the system other than for monitoring purposes. The system will use on-line planning to generate and execute capture trajectories as objects appear on the conveyor, and deliver the objects to their desired goal locations. The delivery of objects may require multiple steps such as regrasps or hand-over of the objects as well as trajectories that avoid obstacles that may move. The system can use both arms cooperatively to manipulate an object with both arms (figure 2), or independently to increase the throughput of the workcell by capturing two objects simultaneously (figure 1), or capturing an object with one arm while the other arm is moving.

Experimental robotic systems with similar objectives have been described by several authors [3, 4], while other authors have focused on developing generic system architectures and their interfaces [7, 2, 9, 19, 8, 11, 13, 12]. Here we will compare ours with a few

representative systems from the view point of the architecture and interfaces.

The KAMRO workcell [4, 5] combines an on-line planner (FATE) and a real-time control subsystem. The input to FATE is a petri-net describing the assembly task. FATE interprets the net and sends elementary commands (transfer, fine-motion, grasp) to the controller, and monitors its progress. Our approach is similar except the commands generated by the planner are *strategic commands* which are further decomposed into elementary operations by a finite-state machine facility *within the control subsystem*. Monitoring and error-recovery occurs in both in the control and planning subsystems. This extra layer allows our system to deal with moving objects and obstacles.

RCS and NASREM [13, 1] are strictly hierarchical approaches to system design. For telerobotic applications, UTAP [7] provides a more detailed specification of individual subsystems and their interfaces. UTAP resembles subsystems-first design in that all modules accept similar messages, however, in UTAP interfaces have state (messages are processed in the context of previous messages), message exchange isn't anonymous (sender specifies receiver), there is limited connectivity (strictly hierarchical messages), and expandability (the type and number of modules is fixed). UTAP's messages are richer because they include language constructs (macros, message groups etc.).

Object-Oriented programming (OOP) facilities are well suited for interfaces-first designs and the construction of modular interfaces. Primitive interfaces can be encapsulated in individual abstract classes, and multiple inheritance used to build dedicated interfaces from the primitive elements. However, most uses of OOP in robotic applications use objects to model the individual subsystems, not the information that needs to be exchanged (subsystems-first approach). For instance, RIPE [8] defines an object hierarchy to describe the different physical elements in the system. In RIPE, class definitions become in effect interface specifications. For instance, the robot class specifies methods to move to a point, move along a certain axis, move along a path, close and open grippers etc. RIPE does not define interfaces for sensor or world state information, nor task specification.

3 Interface Design

The importance of interface design cannot be overstated:

> The greatest leverage in system architecting is at the interfaces. *Eberhardt Rechtin* [15]
>
> The greatest dangers are also at the interfaces *Arthur Raymond*[USC, 1989] [15], pp. 89

3.1 Alternative approaches to system design

Modular system design is a well-established methodology. This approach breaks a large system into smaller, well-defined modules with specified functionality, which then can be developed and tested independently from the others.

While modularity facilitates the development of the individual parts, the need to develop the interfaces [glue] between parts makes the complexity of the overall system much greater than that of the sum of its parts. Hopefully by careful selection of the functional boundaries, the resulting subsystems are somewhat independent and the total complexity can approach the ideal limit of the sum of its parts.

Subsystems-First Traditional modular system design (*subsystems first*) follows the cycle illustrated on the left side of Figure 3. Following the analysis of the complete system, functional units are identified and the system is broken into subsystems across these functional boundaries. These subsystems are further defined by their interfaces to the outside world, which specify their observable behavior. Once the subsystems have been developed and individually tested, custom interfaces (glue logic) are developed to interconnect the different subsystems. This results in a complete system that can now be tested against the design specifications. This process repeats itself on each design iteration.

This *subsystems-first* technique requires that the scope and functionality of the overall system is known in advance. It emphasizes subsystems rather than their interfaces. It is therefore most appropriate for "sparsely-interconnected" one-of-a-kind systems that, once designed, have little need to expand and add new subsystems. For example, a monolithic electronic board (say a graphics board) is designed as a unit from the onset and the different functional units (CPU, video memory, graphic accelerator hardware, etc.) are designed with their own interfaces and require custom glue-logic to be connected to each other within the board. This approach is most appropriate for systems where subsystems have low degree of interconnectivity.

Interfaces-First There are other systems, however, where the full scope isn't known in advance and/or they require a much higher degree of interconnection. These are better designed using the *interfaces-first*

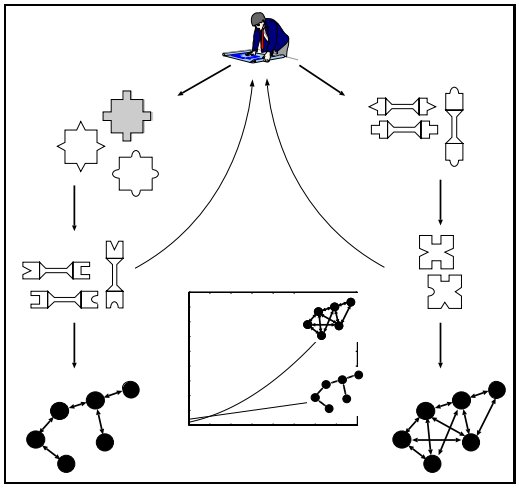

Figure 3: **Contrasted Design Techniques**

Subsystems-first design, on the left, concentrates on developing functional units. Information flow between the units requires custom interfaces. Interfaces-first design on the right starts with then information flow. Functional units are then tailored to the interfaces developed. Interfaces-first design is more appropriate when subsystems are highly interconnected.

methodology, where the types of information flow between the subsystems (whatever they turn out to be) are identified from the onset and interfaces for the information (*not* the subsystems) are designed first. Later, subsystems are designed that use these interfaces to communicate and interact. This design is more typical of large software projects or computer systems. For example the busses in different computer families (e.g. PC bus, VME bus etc.) are designed first, and the different boards that interact through these busses are designed later to conform to the interface specification. In fact, new boards are designed at a later stage with functionality that wasn't anticipated. This design cycle, shown down the right side of Figure 3, is most appropriate for "heavily interconnected" systems and those that are "open-ended" in the sense that new pieces will have to be incorporated at a later stage.

Traditionally, robotic systems have been designed with a subsystems-first technique. That has made them hard to develop and extend due to their limited interconnectivity and the considerable amounts of custom interfacing required.

3.2 Design of robotic interfaces

Design of the subsystem interfaces is not a trivial task for robotic systems because the information/command flow is much more structured (i.e. higher level) and varied than that modeled with a bus-type interface. For example, all the devices connected to a SCSI bus are expected to be able to operate at (almost) the same speed and accept the same type of commands. This isn't a good model when the "devices" are things like graphical interfaces, teleoperator-input devices, planning subsystems, and sensor subsystems. Each of these subsystems requires different fundamental types of information and is able to process it at significantly different speeds.

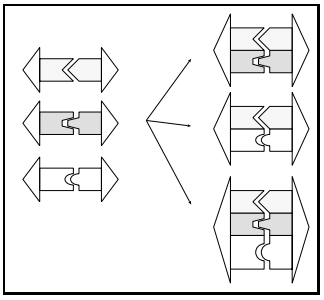

Figure 4: **Modular Interface Design**

First, the fundamental types of information flow are identified, and interfaces designed for each (on left). Subsystem interfaces are then built from combinations of these primitive information interfaces.

We propose to address this problem with a "modular" approach to designing the interfaces themselves. In particular, we first identify the fundamental properties of the information flow. We then divide the information into types based on the characteristics of the flow itself (bandwidth, persistence, idempotency). Next, we design primitive interfaces for each type of information flow. These primitive interfaces can then be combined to create custom "compound" subsystem interfaces. Figure 4 shows this process.

3.3 Primitive interfaces for robotic workcell

In the context of a robotics workcell, we must examine world-model (state) information flow, command flow, control signals, etc., identify parameters that characterize the information, and build primitive interfaces around each information type. This section examines the design of the system architecture for the smart workcell project using this interfaces-first technique.

Three types of information flow can be distinguished in this application: world state, system commands, and high-level task descriptions. Table 1 summarizes the three primitive interfaces and selectable parame-

interface	*nature of the information*	*selectable parameters*
world state interface	periodic, idempotent, unreliable, last-is-best, persistent until the next update	update rate, specific information desired
command interface	asynchronous, reliable, in-order delivery (exactly once), persistent until completed	priority (to resolve conflicting requests)
task interface	asynchronous, persistent until cancelled, reliable but not necessarily in-order	priority

Object information from World Model	
location	object location in the global reference frame.
grasps	possible grasp locations within the object
properties	mass, inertia, limits on acceleration etc.
shape	shape (collision avoidance, graphics)

Robot information from World Model	
location	robot-base location within workcell.
joint val.	value of the joint coordinates (pos, vel, acc)
limits	kinematic (joint) and torque limits
kinematics	Denavit-Hartenberg parameters
state	moving, grasping an object etc.

Robot commands through the command interface	
move object	Move an object that is being grasped. This command will provide a via-point collision-free path for the object. The robot is controlled using object impedance control [17].
move and release	Same as above with the addition of a specified release location (must be close to the end of the trajectory)
move arms	Move one or both arms. The arm(s) to move can't be holding an object. Specifies a via-point collision-free path both in operational and joint space so that the robot controller is free to use different control schemes (and kinematic ambiguities can be resolved).
move and grasp	Same as above with the addition of the specification of the object(s) to be grasped and the corresponding arm(s), and grasp(s) location(s) for each arm involved (any combination of arm/object, including both arms on the same object is allowed)

Task commands through the task interface	
place objects	Place a set of objects at their specified destination. The objects may not even be in the workspace. The system must get the objects however it can, place them at their goal, and monitor to make sure they remain there until the command is cancelled.
make assembly	Similar to place objects, except the object goals are now in contact with each other.

Table 1: Interface characteristics and content

ters for these interfaces, and gives an overview of the specific information that is encapsulated within each interface.

World model state information includes data such as positions and velocities of the different objects (on and off the conveyor), arm locations, joint angles, and force signals. By its very nature this information is either *static* or *periodic* and needs to be updated continuously, because it corresponds to physical quantities that change over time. In addition, the information is *persistent* until it is invalidated by a new update of the same information.

World model state information also has *idempotent* semantics—getting two identical updates of the same information (say the position of an object) causes no side-effect and is logically equivalent to a single update. Moreover, the natural semantics are *last-is-best*, i.e. we are always interested in the most recent value, even at the expense of missing intermediate values. The information content and frequency of each specific instance of the interface must be *customizable* since different subsystems require different subsets of the overall world-model information at widely different rates. For instance, a slow graphical interface animating the robot position may need new joint angles at only 10 Hz; a control or estimator subsystem will need updates at a much higher rate.

System command information is quite different. For one, commands are *asynchronous* and "hold their value" only until they are completed (even if no new command is received). Commands are *not* idempotent and need to be delivered exactly once reliably and in-order. For instance, if we are sending a trajectory to the robot, sending the trajectory twice will cause the robot to execute two motions. Similarly, missing an intermediate command that, say, closed a gripper before lifting an object is not acceptable.

High-level task commands fall somewhat between these two extremes. They are also *asynchronous*. However, tasks are in a sense self-contained (i.e. specify a goal or desired system state) as opposed to a process. As a result they may be *persistent*. For instance, a task may specify to repeatedly pick objects from a conveyor, immerse them in a solution, and then place them on a second conveyor. This task doesn't end until explicitly cancelled. Multiple concurrent tasks may be active at any one time and compete for the system's resources. Tasks may also be built as logical combinations of smaller subtasks (e.g. put object O1 at location L1 and object O2 at location L2).

These three primitive interfaces constitute the information building blocks (modules) from which custom interfaces can be built to connect the different sub-

systems. These interfaces are *anonymous* making no assumptions about the number or the specific subsystems that will be connected through them. Instead, they provide a mechanism for subsystems (whatever they end up being) to share and communicate information. Since they encapsulate the information required to interact and command the robotic workcell, they provide sufficient coverage for subsystem interaction.

On occasion a new subsystem may be developed that requires a different type of information, or a new sensor will be added that provides some other type of information. This will require a new information interface primitive. However, they should not affect the existing interfaces, except for the few new tasks and commands that utilize the new information.

3.4 System architecture

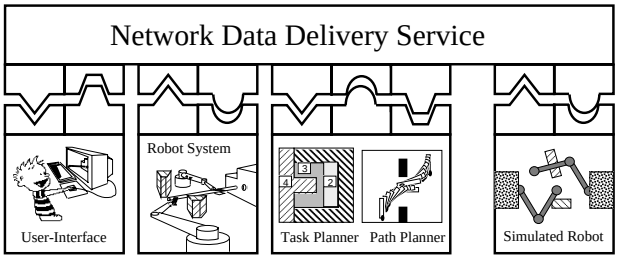

Figure 5: **System Architecture**

Four subsystems collaborate to operate the workcell. Each subsystem uses a combination of the primitive information interfaces to connect to the information flow. The interfaces are built using a "software bus" called the Network Data Delivery Service (NDDS).

The basic system architecture is illustrated in figure 5. The overall system has been broken into four subsystems: Human-interface, robot control system, task and path planners, and the robot simulator. The information primitive interfaces are represented by connection icons: state information (triangle icon), task requests (trapezoid icon), and commands (semicircle icon). The graphical human interface receives state information and provides a graphical representation of the scene to the user. During operation, the high-level task is specified by the human interface and sent to the planners. The task planner/path planner receives continuous updates from the robot and task requests from the user, and produces robot commands to implement the requested tasks. The robot controller executes these commands and processes the sensor signals to create and maintain a world model. The simulator can masquerade as the robot control system during development or, by running it concurrently with

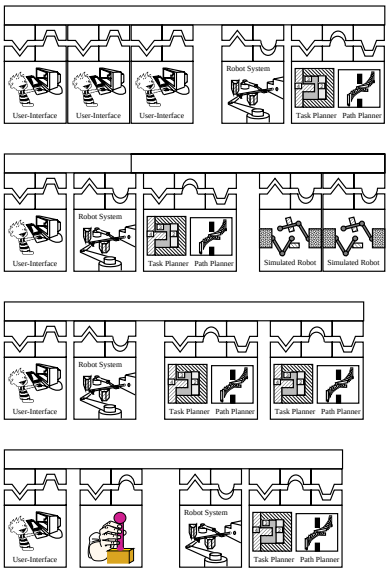

Figure 6: **Software Configurations**

The anonymous interfaces allows replicated modules (top three lines) in the system. The modular information interfaces allow new subsystems to be added to the system (last line).

the robot, can be used to compare the accuracy of the workcell models (kinematics, dynamics and state transitions) with the behavior of the actual system.

The three primitive interfaces are used to communicate between the different subsystems. For instance, the human-interface uses the world-model interface to obtain the kinematics and position of the arms and workspace objects to display them to the user. Tasks are sent to the planners through the task-interface. The planners also use the world-model interface but at a lower bandwidth.

The different subsystems are configured as if connected to a "software bus" where the three primitive interfaces provide the means for the bus-access protocol. This allows functionally identical subsystems to be replicated and new subsystems to be added to create different configurations. The actual implementation of the software bus over the network uses a subscription communications system called the Network Data Delivery System (NDDS) [10, 14].

Information interfaces together with the software bus allow combinations of modules to be easily used. Figure 6 illustrates some of these configurations. In the first three cases, we have replicated modules mentioned earlier. The top line indicates several users collaborating to control the robot, or monitor its activities. The second line the use of multiple simulators

to simulate different aspects of the system, such as for command previewing. The third line shows the use of combined multiple planning strategies, each of which may be more appropriate for certain tasks. Finally, the last line indicates that modular interfaces allow us to build new interfaces such as one for a teleoperation subsystem.

4 System Operation

The workcell can perform autonomous multi-step operations such as the one represented in figure 7. Several points are worth noting: First the planner has continuous access to the state of the workcell though the world-model interface. This state is used both to maintain its internal data structures and performs precomputations, and to monitor completion/failure of issued commands (the stateless nature of the command interfaces precludes any acknowledgments; all feedback to the planner must occur via explicit observation of system behavior). Second most of commands issued by the planner are strategic, requiring the workcell to implement its own event-driven sequencing and monitoring, for instance, the `move and grasp` command is decomposed by the strategic-workcell controller into the following command sequence (figure 8): `approach` (move along planned trajectory), `intercept` (move under a locally generated intercept trajectory that is constantly refined), `track, descend, grasp` (regulate directly from the object position and velocity while descending and closing the grippers), and `lift`. These actions require high-bandwidth world state updates and immediate reactive feedback, and require several control mode switches, see [18] for details. Figure 8, also shows data from the experimental system in operation.

5 Summary and Conclusions

We have presented a novel approach to designing complex robotic systems called *interfaces-first* design. Interfaces-first design first identifies the fundamental types of information flow in the system, and then encapsulates that flow into primitive anonymous interfaces. Subsystem interfaces are then built from combinations of these primitive information interfaces. This approach results in expandable systems with facile interconnectivity.

This design technique has been implemented with a network "software bus", and demonstrated on a dual-arm robot workcell. The workcell is capable of accept-

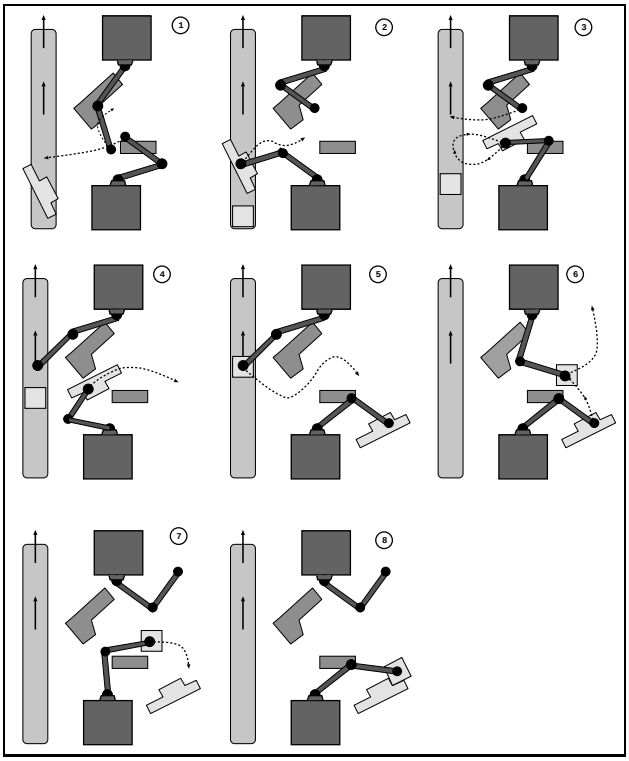

Figure 7: **Typical system operation**

The user has specified the simple assembly of two objects depicted in the bottom-right (8) by dragging the iconic representations of the objects on a graphical interface (not shown). After this the remaining actions are autonomous (left to right, and top to bottom): (1) The planner constantly monitors the workcell using the world-state interface and as soon as a needed object appears on the conveyor, a capture trajectory is planned and sent to the robot [`move and grasp` *command]. This command specifies the top arm to be moved out of the way, and the bottom arm to grasp the object. (2) Once the object is grasped, the planner (which detects that through the world-state interface) plans a delivery trajectory and sends it [*`move and release` *command]. The object is placed at an intermediate location where the arm can change handedness. (3) In the meantime, a new object has appeared on the conveyor, so the planner issues a* `move and grasp` *command for both arms (one for the conveyor object, the other for the just-released object). (4) While one arm picks the second object from the conveyor, the other delivers the first object to its destination [*`move and release` *command]. (5) Once the second object is grasped, since the final destination is only reachable by the other arm, the grasping arm is commanded to place it at a location reachable by both arms [*`move and release`*]. (6) Next a* `move and grasp` *command moves the first arm away while the second picks the object, and finally, (7) a* `move and release` *command delivers the object to its final destination (8).*

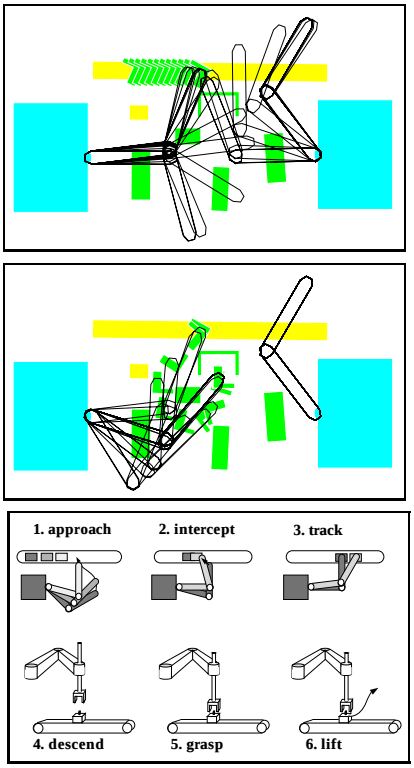

Figure 8: **Animation of system operation**

The top two figures are animations at 0.8 sec intervals of data collected during the capture (`move-and-grasp`*) and delivery (*`move-and-release`*) of an object. The bottom figure illustrates the stages orchestrated by the strategic controller during the* `move-and-grasp` *command.*

ing high-level user task requests, planning motions to carry them out, acquiring objects from a moving conveyor, and performing simple assemblies.

Acknowledgements

This work was supported under ARPA grant N00014-92-J-1809.

References

[1] J. S. Albus, H. G. McCain, and R. Lumia. NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM). Technical report, NIST, 1989.

[2] S. Fleury, M. Herrb, and R. Chatila. Design of a Modular Architecture for Autonomous Robot. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3508–13, San Diego, CA, May 1994.

[3] Li-Chen Fu and Yung-Jen Hsu. Fully automated two-robot flexible assembly cell. In *IEEE International Conference on Robotics and Automation*, pages 332–338, Atlanta, Georgia, USA, May 1993.

[4] A. Hormann and U. Rembold. Development of an Advanced Robot for Autonomous Assembly. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2452–57, Sacramento, CA, May 1991.

[5] Andreas Hormann. On-Line Planning of Action Sequences for a Two-Arm Manipulator System. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1109–1114, Nice, France, May 1992.

[6] Tsai-Yen Li and Jean-Claude Latombe. On-Line Motion Planning for a Two-Arm Manufacturing Workcell. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 1, Nagoya, Japan, May 1995.

[7] R. Lumia, J. L. Michaloski, R. Russel, T. E. Wheatley, P. G. Backes, S. Lee, and R. D. Steele. Unified Telerobotic Architecture Project (UTAP) Interface Document. Technical report, NIST, Intelligent Systems Division, NIST, Gaithersburg, MD, June 18 1994.

[8] D.J. Miller and R.C. Lennox. An Object-Oriented Environment for Robot System Architectures. *IEEE Control Systems Magazine*, 11(2):14–23, February 1991.

[9] F. R. Noreils. Toward a robot architecture integrating cooperation between mobile robots. *The International Journal of Robotics Research*, 12(1):79–98, February 1993.

[10] G. Pardo-Castellote and S. A. Schneider. The Network Data Delivery Service: Real-Time Data Connectivity for Distributed Control Applications. In *Proceedings of the International Conference on Robotics and Automation*, San Diego, CA, May 1994. IEEE.

[11] D. W. Payton, J. K. Rosenblatt, and D. M. Keirsey. Plan guided reaction. *IEEE Transactions on Systems, Man and Cybernetics*, 20(6):1370–82, Nov-Dec 1990.

[12] Judson P. Jones Philip L. Butler. A Modular Control Architecture for Real-Time Synchronous and Asynchronous Systems. In *Proceedings of the SPIE - Applications of Artificial Intelligence: Machine Vision and Robotics*, volume 1964, pages 287–298. SPIE, 1993.

[13] R. Quintero and A.J. Barbera. A Real-Time Control System Methodology for Developing Intelligent Control Systems. Technical Report NISTIR 4936, NIST, October 1992.

[14] Real-Time Innovations, Inc., 954 Aster, Sunnyvale, CA 94086. *NDDS: The Network Data-Delivery Service User's Manual*, 1.7 edition, November 1994.

[15] Eberhardt Rechtin. *Systems Architecting: Creating and Building Complex Systems.* Prentice-Hall, first edition, 1991.

[16] S. Schneider. *Experiments in the Dynamic and Strategic Control of Cooperating Manipulators.* PhD thesis, Stanford University, Stanford, CA 94305, September 1989. Also published as SUDAAR 586.

[17] S. Schneider and R. H. Cannon. Object Impedance Control for Cooperative Manipulation: Theory and Experimental Results. *IEEE Transactions on Robotics and Automation*, 8(3):383–94, June 1992.

[18] S. A. Schneider, V. Chen, and G. Pardo-Castellote. Object-Oriented Framework for Real-Time System Development. In *Proceedings of the International Conference on Robotics and Automation*, Nagoya, Japan, May 1995. IEEE.

[19] F. Zanichelli, S. Caselli, A. Natali, and A. Omicini. A Multi-Agent Framework and Programming Environment for Autonomous Robotics. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3501–7, San Diego, CA, May 1994.