UPE: Utah Prototyping Environment for Robot Manipulators

Mohamed Dekhil, Tarek M. Sobh, Thomas C. Henderson, and Robert Mecklenburg*

Department of Computer Science
University of Utah
Salt Lake City, Utah 84112

Abstract

Developing an environment that enables optimal and flexible design of robot manipulators using reconfigurable links, joints, actuators, and sensors is an essential step for efficient robot design and prototyping. Such an environment should have the right "mix" of software and hardware components for designing the physical parts and the controllers, and for the algorithmic control of the robot modules (kinematics, inverse kinematics, dynamics, trajectory planning, analog control and digital computer control). Specifying object-based communications and catalog mechanisms between the software modules, controllers, physical parts, CAD designs, and actuator and sensor components is a necessary step in the prototyping activities. In this paper, we propose a flexible prototyping environment for robot manipulators with the required subsystems and interfaces between the different components of this environment.

1 Introduction

Prototyping is an important activity in engineering. Prototype development is a good test for checking the viability of a proposed system. Prototypes can also help in determining system parameters, ranges, or in designing better systems. The interaction between several modules (e.g., S/W, VLSI, CAD, CAM, Robotics, and Control) illustrates an interdisciplinary prototyping environment that includes radically different types of information, combined in a coordinated way.

The goal of this research project is to build a framework for optimal and flexible design of robot manipulators with the necessary software and hardware systems and modules. This framework is composed of several subsystems such as: *optimal design, simulation, control, monitoring, CAD/CAM modeling, part ordering, and physical assembly and testing.* Each subsystem has its own structure, data representation, and reasoning strategy. On the other hand, much of the information is shared among these subsystems. To maintain the consistency of the whole system, an interface layer is proposed to facilitate the communication between these subsystems, and set the protocols that enable the interaction between the subsystems to take place. Figure 1 shows a schematic view of the proposed prototyping environment with its subsystems and interface.

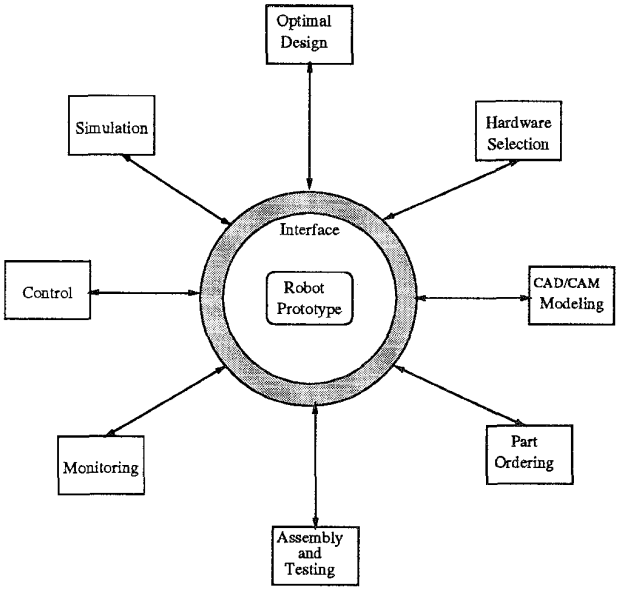

Figure 1: The prototyping environment.

2 Background

To integrate the work among different teams and sites working in such a large project, there must be some kind of synchronization to facilitate the communication and cooperation between them. A concurrent engineering infrastructure that encompasses multiple sites and subsystems, called Palo Alto Collaborative Testbed (PACT), was proposed in [2]. The issues discussed in that work were: cooperative development of interfaces, protocols, and architecture, sharing of knowledge among heterogeneous systems, and computer-aided support for negotiation and decision-making.

An execution environment for heterogeneous systems called "InterBase" was proposed in [1]. It integrates preexisting systems over a distributed, autonomous, and heterogeneous environment via a tool-based interface. In this envi-

*This work was supported in part by DARPA grant N00014-91-J-4123, NSF grant CDA 9024721, and a University of Utah Research Committee grant. All opinions, findings, conclusions or recommendations expressed in this document are those of the author and do not necessarily reflect the views of the sponsoring agencies.

IEEE International Conference
on Robotics and Automation
0-7803-1965-6/95 $4.00 ©1995 IEEE

ronment each system is associated with a *Remote System Interface (RSI)* that enables the transition from the local heterogeneity of each system to a uniform system-level interface.

Object orientation and its applications to integrate heterogeneous, autonomous, and distributed systems are discussed in [7]. The argument in this work is that object-oriented distributed computing is a natural step forward from the client-server systems of today. Automated, flexible and intelligent manufacturing based on object-oriented design and analysis techniques is discussed in [6], and a system for design, process planning and inspection is presented.

A management system for the generation and control of documentation flow throughout a whole manufacturing process is presented in [5]. The method of quality assurance is used to develop this system that covers cooperative work between different departments for documentation manipulation.

A computer-based architecture program called *the Distributed and Integrated Environment for Computer-Aided Engineering* (Dice), which addresses the coordination and communication problems in engineering, was developed at the MIT Intelligent Engineering Systems Laboratory [8].

In the environment we are proposing, several subsystems are communicating through a *central interface layer* (CI), and each subsystem has a *subsystem interface* (SSI) responsible for data transformation between the subsystem and the CI. Adding new subsystem can be achieved by writing an SSI for this new subsystem, adding it to the list of the subsystems in the CI, and no changes required to the other SSIs. Removing a subsystem only requires removing its name from the subsystems list in the CI.

3 The Prototyping Environment

The proposed environment consists of several subsystems each of which carry out certain tasks to build the prototype robot. These subsystems share many parameters and information. To maintain the integrity and consistency of the whole system, a central interface (CI) is proposed with the required rules and protocols for passing information. This interface is the layer between the robot prototype and the subsystems, and it also serves as a communication channel between the different subsystems.

3.1 Overall Design

The Prototyping Environment (PE) consists of a *central interface* (CI) and *subsystem interfaces* (SSI). The tasks of the central interface are to:

- Maintain a global database of all the information needed for the design process.
- Communicate with the subsystems to update any changes in the system. This requires the central interface to know which subsystems need to know these changes and send messages to these subsystems informing them of the required changes.
- Receive messages and reports from the subsystems when any changes are required, or when any action has been taken (e.g., update complete).
- Transfer data between the subsystems upon request.
- Check constraints and apply some of the update rules.
- Maintain a design history containing the changes and actions that have been taken during each design process with date and time stamps.
- Deliver reports to the user with the current status and any changes in the system.

The subsystem interfaces are the interface layers between the CI and the subsystems. This makes the design more flexible and enables us to change any of the subsystems without much change in the CI — only the corresponding SSI need to be changed. The role of the SSIs are:

- Report any changes to the CI.
- Receive messages from the CI with required updates.
- Perform the necessary updates in the actual files of the subsystem.
- Send acknowledgments or error messages to the CI.

The assumption is that there is a user at each subsystem (by a user here we mean one or more skilled persons who understand this subsystem), and there is a user operating the central interface as a general director and coordinator for the design process. In other words, the CI is to assist in the coordination of the job and to help communicate with all subsystems. Figure 2 shows an overall view of the suggested design.

In the first phase of implementing the PE, the users have more work to do. The CI and SSIs maintain the information routing between the subsystems by sending messages to the corresponding user at each subsystem, then the action itself (e.g., update a file) is accomplished by the user. Later on, the system will be automated to perform most of these actions itself and the user will simply be informed of the actions taken.

3.2 Communication Protocols

The main purpose of this environment is to keep all the subsystems informed of any changes in the design parameters. Therefore, passing information between the subsystems is the most important part of this environment. To be able to control the information flow, some protocols were developed to enable the communication between these subsystems in an organized manner. In our design, all subsystems communicate through the CI which is responsible for passing the information to the subsystems that need to know.

There are two types of events that can occur in this system:

1. Change reported from one of the subsystems.
2. Request for data from one subsystem to another.

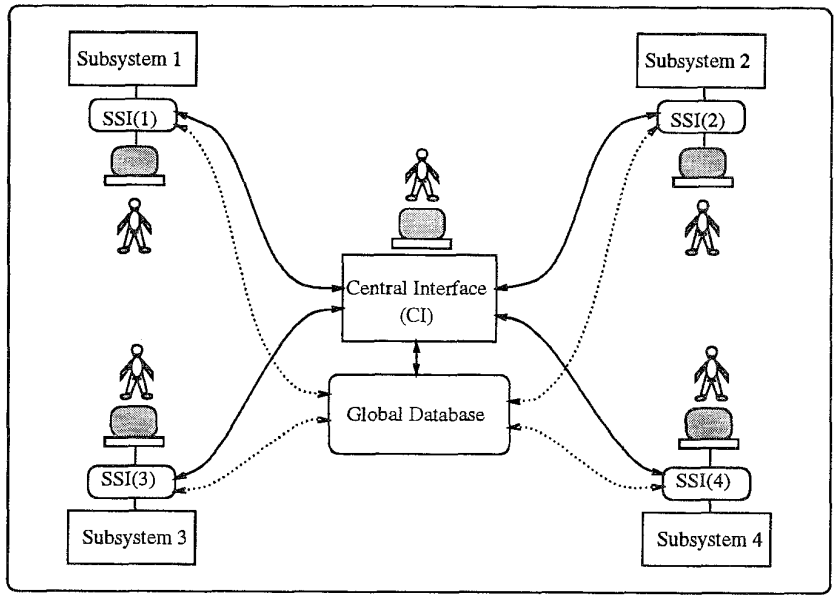

Figure 2: Overall design of the prototyping environment.

Figure 3 shows the protocol used for the first event represented by a finite state machine (FSM). The states of this FSM are:

1. Steady state: Do nothing.
2. Change has been reported: send lock message to all subsystems. Apply relations and check constraints. If constraints are satisfied, go to state 3. If constraints are not satisfied, report these to sender and go to steady state.
3. Constraints are satisfied: Notify the subsystems with the changes and wait for acknowledgments.
4. Acknowledgments received from all subsystems: Send the final acknowledgment to the subsystems and go to steady state.
5. Acknowledgments not Ok: Send a "change-back" command to the subsystems and go to steady state.

Figure 4 shows the protocol for the second event. The states in this FSM are:

1. Steady state: Do nothing.
2. Request for SS2 received from SS1. Send the request to SS2.
3. Required data found at SS2. Send data to SS1 and go to steady state.
4. Required data not found at SS2. Send report to SS1 and go to steady state.

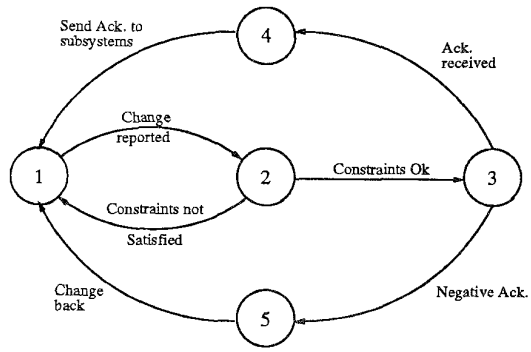

Figure 3: The change-parameter protocol.

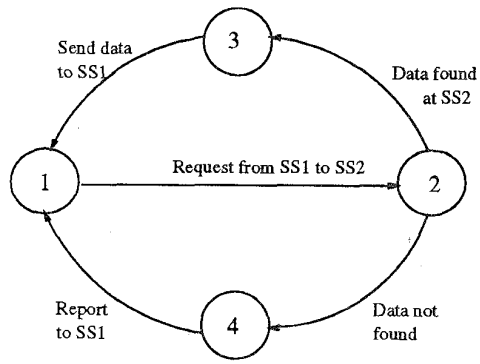

Figure 4: Data request protocol.

3.3 Prototyping Environment Database

A database for the system components and the design parameters is necessary to enable the CI to check the constraints, to apply the update rules, to identify the subsystems that should be informed when any change happens in the system, and to maintain a design history and supply the required reports.

This database contains the following:

- Robot configuration.
- Design parameters.
- Available platforms.
- Design constraints.
- Subsystems information.
- Update rules.

Now the problem is to maintain this database. One solution is to use a database management system (DBMS) and integrate it in the prototyping environment. This requires writing an interface to transform the data from and to this DBMS, and this interface might be quite complicated. The other solution is to write our own DBMS. This sounds difficult, but we made it very simple since the amount of data we have is limited and does not need sophisticated mechanisms to handle it. A relational database model is used in our design, and a user interface has been implemented to maintain this database. For the current design, by making a one-to-one correspondence between the classes and the files, reading and writing a file can be accomplished by adding member functions to each class.

3.4 Constraints and Update Rules Compiler

A compiler is provided to generate C++ code for the constraints and the update rules. First, the syntax of the language that is used to describe the constraints and the update rules is described. Second, the generated code is determined. Using a compiler instead of generic on-line evaluator for the constraints and the update rules has the following advantages:

- All constraints are saved in one text file (likewise the update rules). This makes the data entry very easy. We can add, update, and delete any constraint or update rule using any text editor.
- Complicated data structures are not required for evaluation.
- The database is very simple, which facilitates maintaining the design history.
- Format changes, or changes in the generated code require only changes to the compiler, and no changes in the system are required.

On the other hand, it has the following disadvantages:

- The generated code has to be included in the system and the whole system must be recompiled.
- A compiler needs to be implemented.

By analyzing the design constraints and the update rules, we constructed a simple description of the language to be input to the compiler. There are two options in this design, either to have one compiler for both the constraints and the rules, or to build two compilers, one for each. From the analysis of the constraints and the rules we found that there are many similarities between them; thus building one compiler for both is the logical option in this case.

A complete language definition in Backus Naur Form (BNF) along with some examples can be found in [3].

3.5 The Generated Code

As mentioned before, this compiler generates C++ code which is integrated with the CI system to check the constraint or apply the update rule. Each variable in the input to the compiler corresponds to one design parameter. For example, "link1_length" corresponds to the variable in the CI system that represents the length of link number one in the robot configuration. The code generator uses a lookup table to find the corresponding variable name, and this table is part of the CI database.

To update the constraints or the update rules the file containing the old definition will be displayed and the user can add, delete, or update any of the old definitions. Then the new file will be compiled and integrated with the system.

4 Implementation

In the following subsections some implementation issues are investigated, and the different components in our design and how we implemented each of them are described.

4.1 The Central Interface

The central interface (CI) is the core program that handles the communication between the subsystems, and maintains a global database for the current design and a history of previous designs.

The CI is the implementation of the communication protocols described in Section 3.2. Some features and enhancement to the protocols have been added to the CI. For example, when the CI receives a *change* message from an SSI, it directly sends lock messages to the other subsystems so that no more changes can be sent from any SSI until they receive a *steady* message. This solves the concurrency problem if more than one system send changes to the CI at the same time. The first message received by the CI will be handled and the others will be ignored. If an SSI receives a *lock* message after it sent a *change* message, that means its message was ignored. Another feature added to the CI is the ability to detect if an SSI is working or not by tracing the *SSI_Start* and *SSI_Stop* messages.

4.2 The PE Control System

The CI as described above has no user interface. To be able to control and manage the coordination between the subsystems, the PE control system (PECS) was implemented with some functionalities that enable the user to have some control over the CI.

The PECS is built on top of a simple DBMS and a simple compiler for the update rules and the constraints. The user specifies the constraints and/or the update rules using a certain format (a language), then the compiler transforms this to C code that is integrated with the system for constraint checking, and for applying the update rules. The compiler consists of two parts, a parser and a code generator. In the first phase the complexity of the compiler was reduced by making the user language less sophisticated.

4.3 Initial Implementation of the SSIs

In the first phase of implementation, the SSIs serve as a simple interface layer between the CI and the user at each subsystem. They receive messages from the CI and display them to the user who takes any necessary actions. They also report any changes to the CI, and this is done by sending a message to the CI with the changes.

In the next implementation phase, some of the actions will be automated and the user at each subsystem will be notified with any action taken. For example, updating a data file that is used by the subsystem can be automatically done by the SSI, given that it has the necessary information about the file format and the location of the changed data.

4.4 The Central Interface Monitor

The central interface monitor (CIM) enables the user to monitor the actions and the messages passing between the CI and the SSIs with a graphical interface. This interface shows the CI in the middle and the SSIs as small boxes surrounding the CI. The CIM also has a small text window at the bottom. This text window displays a text describing the current action. The messages are represented by an arrow from the sender to the receiver.

5 Results

One of the test cases for the prototyping environment is shown in Figure 5. In this case, the optimal design subsystem changed the length of one of the robot links and sent a data-change message to the CI. The CI in turn sent lock messages to all other subsystems notifying them that no changes will be accepted until they receive a final acknowledgment message. Then, the CI applied the relations and checked the design constraints. In this test case the constraints were satisfied, so the CI sent these changes to the subsystems that needed to be notified. After that, the CI waited for acknowledgments from the subsystems. In this case it received positive acknowledgments from the specified subsystems. Finally, the CI updated the database and sent final acknowledgment messages to all subsystems. More results and test cases can be found in [4].

6 Conclusion

The design basis for building a prototyping environment for robot manipulators was investigated and the design options were explained. An initial implementation of a central interface and some of the subsystem interfaces was done to demonstrate the functionality of the proposed environment. The design constraints and the update rules are expressed using simple syntax and are saved as part of the environment database. A graphical user interface to control and monitor the activities of the environment was implemented. A three-link robot manipulator was built to explore the basis of building this environment. This prototype robot will be used as an educational tool in control and robotics classes. We believe this framework will facilitate and speed the design process of robot manipulators.

References

[1] BUKHRES, O. A., CHEN, J., DU, W., AND ELMAGARMID, A. K. Interbase: An execution environment for heterogeneous software systems. *IEEE Computer Magazine* (Aug. 1993), 57–69.

[2] CUTKOSKY, M. R., ENGELMORE, R. S., FIKES, R. E., GENESERETH, M. R., GRUBER, T. R., MARK, W. S., TENENBAUM, J. M., AND WEBER, J. C. PACT: An experiment in integrating concurrent engineering systems. *IEEE Computer Magazine* (Jan. 1993), 28–37.

[3] DEKHIL, M. Prototyping environment for robot manipulators. Master's thesis, University of Utah, Salt Lake City, UT 84112, Mar. 1994.

[4] DEKHIL, M., SOBH, T. M., HENDERSON, T. C., AND MECKLENBURG, R. Robotic prototyping environment (progress report). Tech. Rep. UUCS-94-004, University of Utah, Feb. 1994.

[5] DUHOVNIK, J., TAVCAR, J., AND KOPOREC, J. Project manager with quality assurance. *Computer-Aided Design 25*, 5 (May 1993), 311–319.

[6] MAREFAT, M., MALHORTA, S., AND KASHYAP, R. L. Object-oriented intelligent computer-integrated design, process planning, and inspection. *IEEE Computer Magazine* (Mar. 1993), 54–65.

[7] NICOL, J. R., WILKES, C. T., AND MANOLA, F. A. Object orientation in heterogeneous distributed computing systems. *IEEE Computer Magazine* (June 1993), 57–67.

[8] SRIRAM, D., AND LOGCHER, R. The MIT dice project. *IEEE Computer Magazine* (Jan. 1993), 64–71.

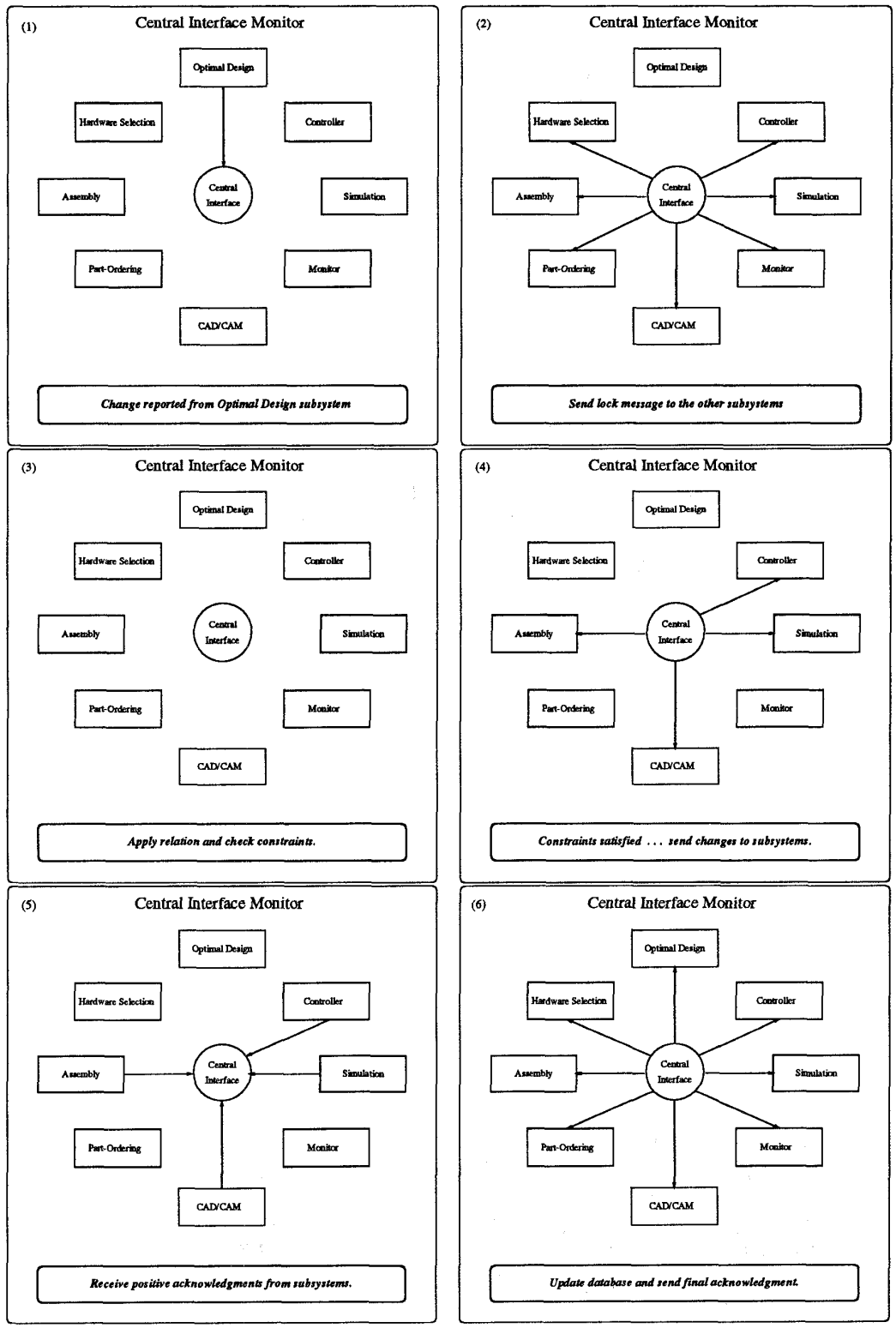

Figure 5: CI test case, success case for data change.