

Appendix C. Robot Controller System Case Study

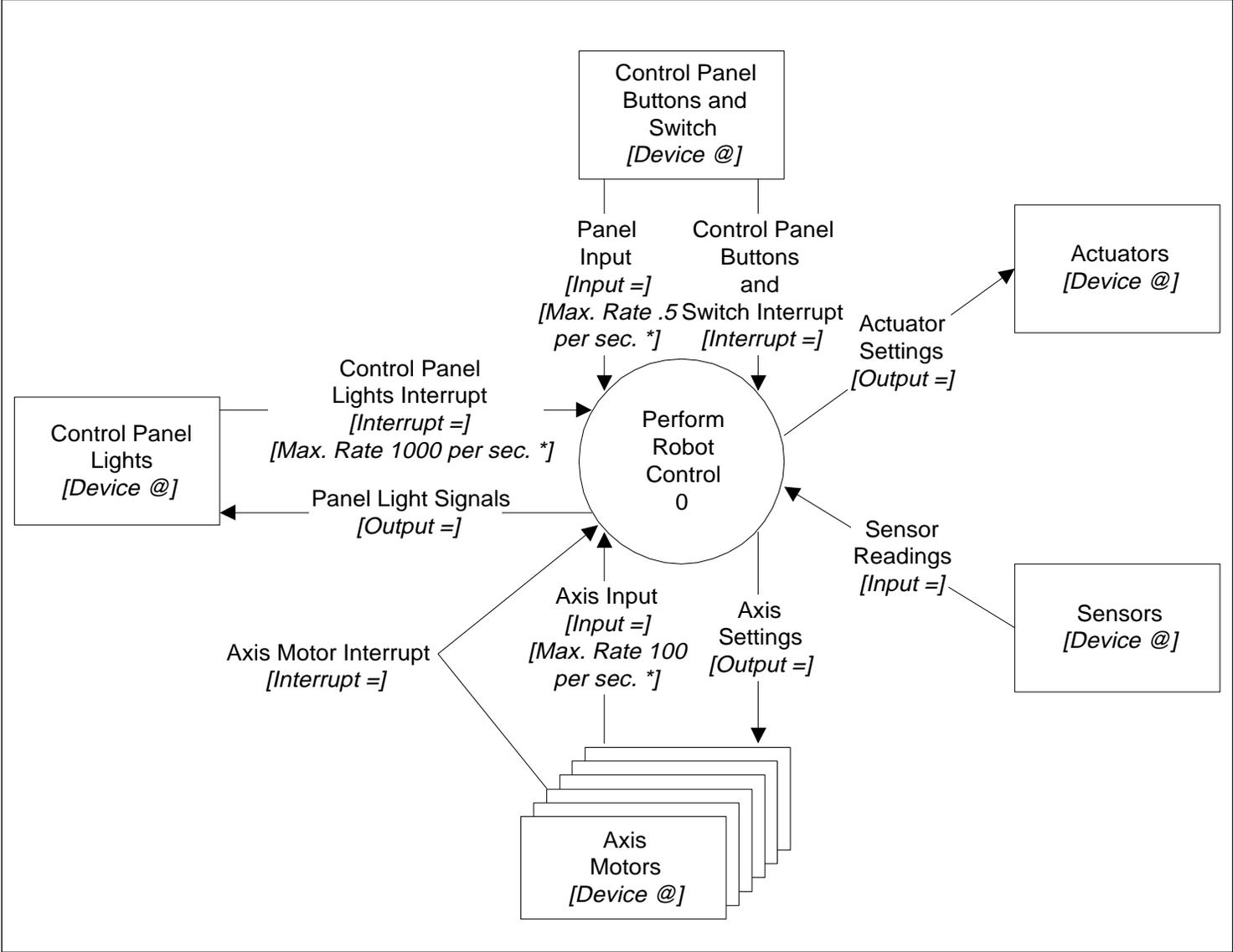
This appendix describes an application of the proof-of-concept prototype, CODA, described in Chapter 10, to a robot controller system. The specification for this system consists of a set of hierarchically arranged data/control flow diagrams, one state-transition diagram, and a textual description. The specification, taken from Gomaa.¹ [Gomaa93, Chapter 23], uses RTSA without COBRA restrictions. From this specification, CODA generates, with the help of an experienced designer, two designs. One design uses the default target environment description, while the second design uses a target environment description where no message queuing services are available. This illustrates how multiple designs can be generated from the same input specification. The case study then shows how a small change to the specification, a change that aligns the specification more closely with COBRA restrictions on the use of RTSA notation, can reduce the number of consultations that CODA must take with the designer, while still leading to the same design.

C.1 Robot Controller System, Version 1

Figure 51 shows the context diagram for Gomaa's robot controller system. As in the previous case study, see Appendix B Table 31, the context diagram is annotated with information inferred or elicited as a result of applying CODA to analyze the specification.

¹ Another treatment of the same robot controller problem is given by Nielsen and Shumate. [Nielsen88]

Figure 51. Annotated Context Diagram for Robot Controller System



The context diagram differs from Gomaa's context diagram in only two ways. First, events arriving from external devices are shown in Figure 51 as dashed, directed arcs. These events are not shown in Gomaa's context diagram, but can be inferred to exist from reading the accompanying textual specification. Second, the six axis motors that control the robot arm are depicted explicitly in Figure 51. Gomaa's context diagram shows a single axis motor but the accompanying textual specification indicates that six axis motors exist.

C.1.1 Analyzing the Specification

Once the data/control flow diagram hierarchy is flattened, the entire data/control flow diagram for the robot controller system consists of twenty-six nodes (18 transformations, 5 terminators, and 3 data stores) and 48 arcs (28 data flows and 20 event flows). The designer asks CODA to classify concepts on the data/control flow diagram, to elicit any additional information required, and to verify that concepts within the specification satisfy appropriate axioms.

C.1.1.1 Classifying Concepts in the Specification

Initially, CODA consults with the designer about the nature of the terminators. After the designer indicates that all terminators in the specification are devices, concept classification proceeds without further consultation until stage four, where CODA requires additional information in order to determine a classification for four data transformations. In one case, CODA needs to know whether a triggered function, Change Program, completes during the triggering transition. In a second case, CODA asks the

designer to confirm, or override, a tentative classification for a transformation, Process Actuator Command, as a synchronous function. In the remaining two cases, Process Motion Command and Receive Acknowledgement, CODA requests that the designer provide information about the time required to execute each transformation. CODA considers two factors. First, will the time, no matter how brief, required to execute a transformation unduly delay an invoking transformation? This might occur, for example, if the invoking transformation receives external events that could be missed while waiting for the invoked transformation to complete. A second factor, considered only if the first factor is not an issue, gauges the amount of time needed to complete the algorithm embodied within a transformation. If substantial time is needed to execute the algorithm, then CODA classifies the transformation as an asynchronous function. In the robot controller case study, the designer indicates that neither of the two transformations, Process Motion Command and Receive Acknowledgement, unduly delay the invoking transformation nor require substantial execution time. From these additional facts, CODA classifies the transformations as synchronous functions.

C.1.1.2 Eliciting Additional Information and Verifying Concepts

Once concept classification is completed, CODA determines that several additional facts are needed about some of the newly classified concepts. Two timers require periods and three asynchronous inputs or outputs require maximum rates. CODA then asks the designer for any specification addenda. In this case study, the designer specifies no addenda. CODA does not ask the designer about locked-state events because

this addendum was included when the input specification was entered with the text editor. Only one event, Ended, see the state-transition diagram in Figure 54, was specified as a locked-state event. This addendum was entered ahead of time to show that CODA does not attempt to elicit information that already exists. The designer finishes adding information to the specification by changing the cardinality of the Axis Controller from one to six. Next, the designer asks CODA to verify that all concepts are classified completely and that all axioms are satisfied. CODA determines that classifications are complete and that axioms are satisfied.

C.1.1.3 Annotated Data/Control Flow Diagram

The results from analyzing the specification are presented in the form of an annotated data/control flow diagram, shown in Figures 52 and 53, for the robot controller system. Figure 52 shows the initial decomposition of the system transformation, Perform Robot Control, from the context diagram. This decomposition differs from that provided by Goma in only one detail. Goma depicts a single transformation, Process Sensor/Actuator Command, which is shown in Figure 52 as two, distinct transformations, Process Actuator Command and Process Sensor Command. This decomposition allows an illustration of the use of labels on data flows to and from data stores. In addition, this decomposition more accurately reflects the processing described in the textual specification for the robot controller system.

The classifications assigned and the additional information elicited by CODA are shown in the annotations for each specification element in Figure 52. Only one

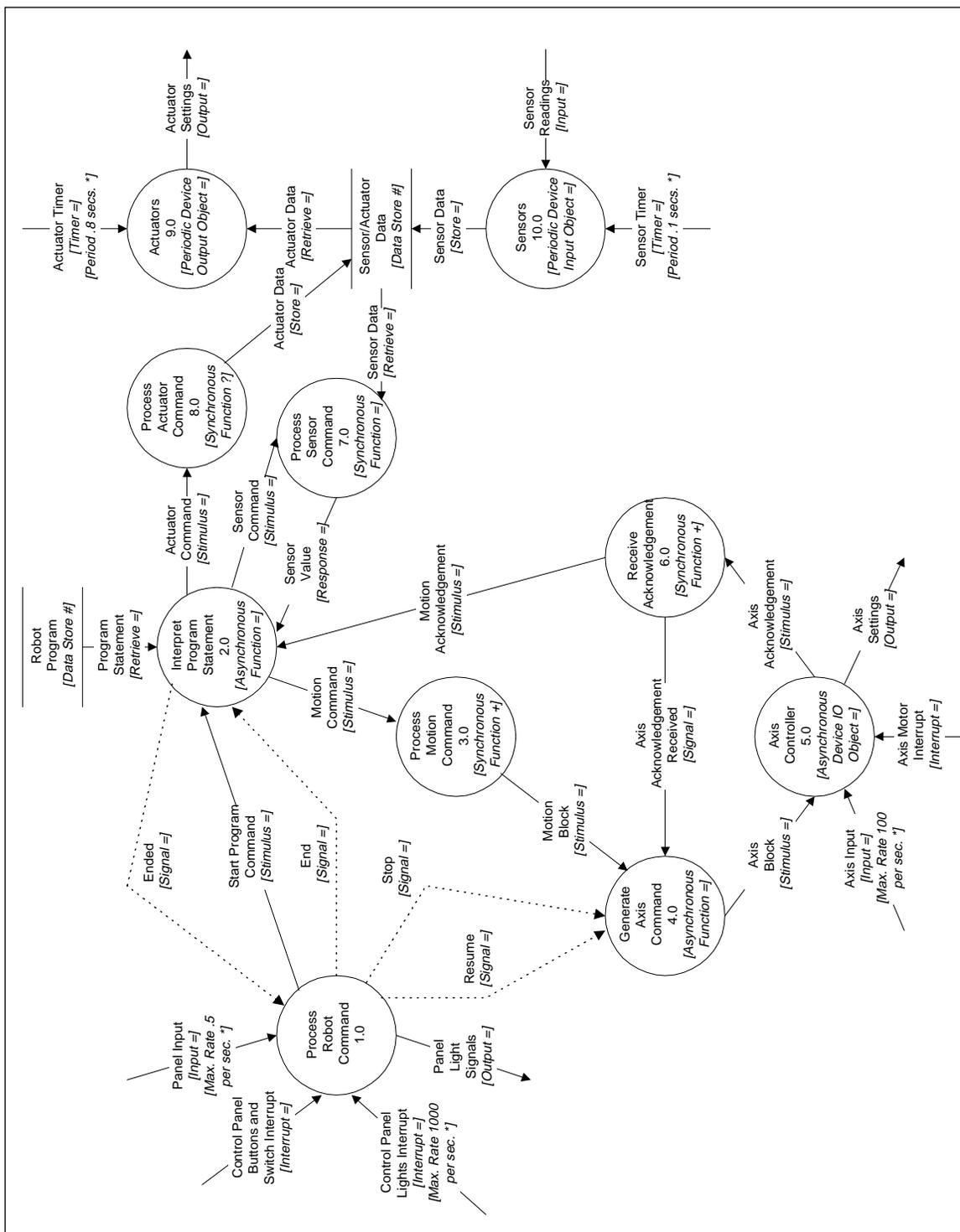


Figure 52. Initial Decomposition of the Robot Controller System

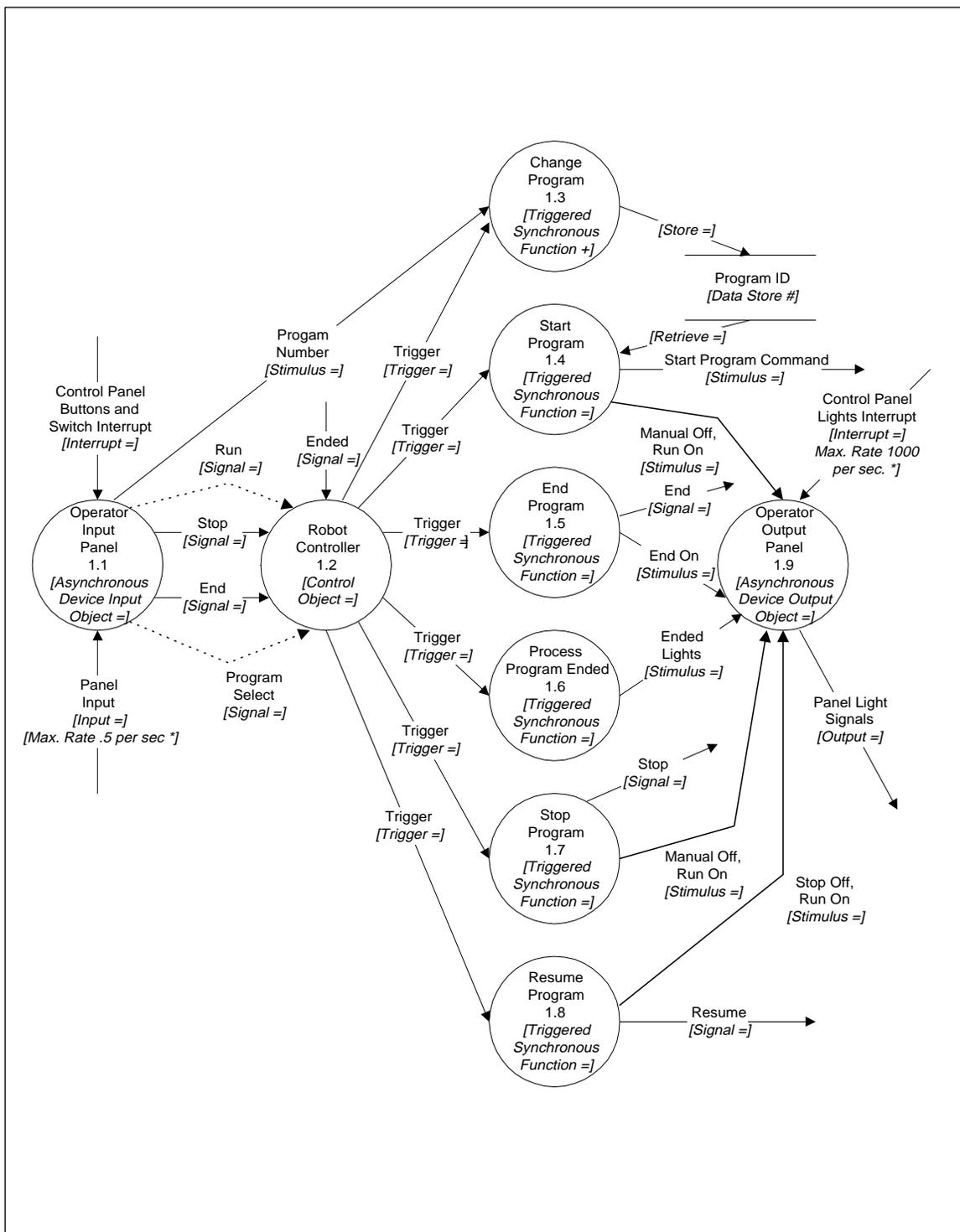


Figure 53. Decomposition of Process Robot Command

transformation, Process Robot Command, remains unannotated. Process Robot Command is further decomposed in Figure 53. This decomposition is identical to that provided by Goma. Each specification element on Figure 53 is annotated. The control transformation, Robot Controller, encapsulates a state-transition diagram, Figure 54.

C.1.2 Generating the Design

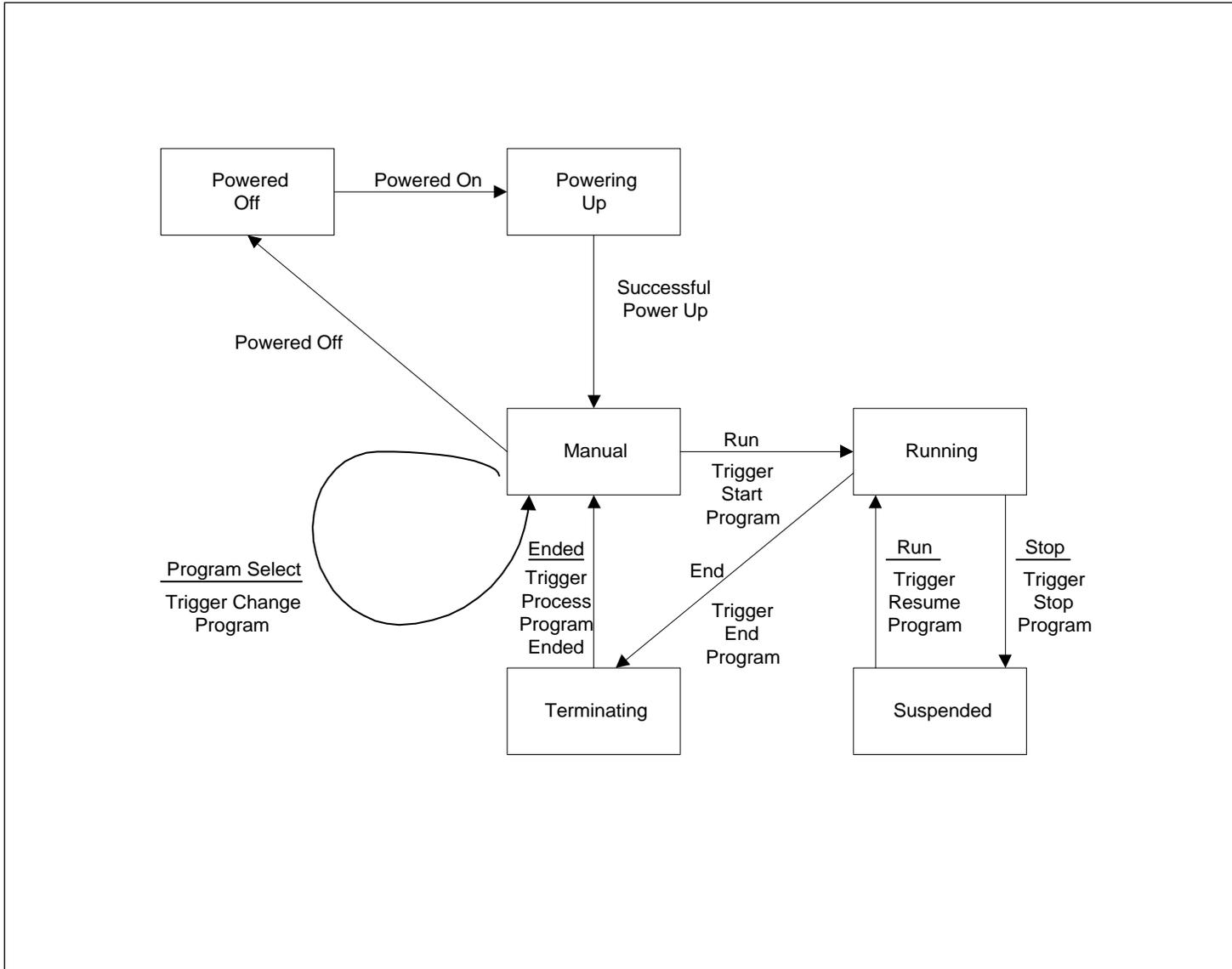
After analyzing the specification, the designer decides to generate a design. Since the designer is experienced, CODA can interact with the designer as necessary to request guidance. The designer chooses first to structure tasks.

C.1.2.1 Structuring Tasks

Whenever a synchronous function within a data/control flow diagram connects with transformations that have been allocated to separate tasks, CODA cannot determine whether the synchronous function is more cohesive with some of these transformations than with others. CODA asks the designer for advice in each such situation; but the designer is not forced to provide guidance. In case of the robot controller system, CODA consults the designer regarding the allocation of two data transformations.

One synchronous function, Process Motion Command, connects two asynchronous functions, Interpret Program Statement and Generate Axis Command. No clear information exists to allow CODA to decide that Process Motion Command should be allocated together with either of the connected transformations; thus, CODA seeks guidance from the designer. The designer, understanding the application-specific relationships among these transformations, indicates that Process Motion Command

Figure 54. State-Transition Diagram for Robot Controller



should be allocated together with Interpret Program Statement, rather than with Generate Axis Command. Had the designer not known how to allocate Process Motion Command, then CODA would generate a separate task for the transformation.

A second case where CODA seeks guidance from the designer involves the synchronous function named Receive Acknowledgement. This function links three adjacent transformations, Axis Controller, Generate Axis Command, and Interpret Program Statement, that are each allocated to a distinct task. In this case, the designer understands, and indicates, that Receive Acknowledgement should be allocated together with Generate Axis Command. Absent guidance from the designer, CODA would simply allocate Receive Acknowledgement to a separate task. Table 45 gives the results of CODA's task structuring including: the tasks created, the transformations allocated to each task, and the criterion used in determining each allocation.

C.1.2.2 Structuring Modules

Next, the designer decides to structure the transformations and data stores into information hiding modules. Here, the same two synchronous functions, Process Motion Command and Receive Acknowledgement, which caused CODA to consult with the designer during task structuring also require consultation during module structuring. The synchronous functions in question each link two other functions that are already allocated to two, distinct modules. For each of the functions in question, CODA cannot determine

Table 45. Task Structuring Decisions for Robot Controller, Version 1

Task	Transformations	Structuring Criterion
Axis Manager	Generate Axis Command Receive Acknowledgement	Asynchronous Internal Task User-Specified Cohesion
Interpreter	Interpret Program Statement Process Sensor Command Process Actuator Command Process Motion Command	Asynchronous Internal Task Sequential Cohesion Sequential Cohesion User-Specified Cohesion
Robot Command Processor	Robot Controller Resume Program Stop Program Process Program Ended End Program Start Program Change Program	Control Task Control Cohesion Control Cohesion Control Cohesion Control Cohesion Control Cohesion Control Cohesion
Process Actuator Output	Actuators	Periodic Device I/O Task
Process Sensor Input	Sensors	Periodic Device I/O Task
Axis Controller	Axis Controller	Asynchronous Device I/O Task
Control Panel Input Handler	Operator Input Panel	Asynchronous Device I/O Task
Control Panel Output Handler	Operator Output Panel	Asynchronous Device I/O Task

whether to allocate the function to one or the other of the two existing modules or to create a new module for the function. CODA consults the experienced designer for guidance. In this case, the designer indicates that the two transformations in question should each be allocated to an existing module. The designer allocates Process Motion

Command to the same module as Interpret Program Statement and allocates Receive Acknowledgement to the same module as Generate Axis Command. Had the designer not provided help, then CODA would have simply generated a separate module for each of the transformations in question. CODA makes the remainder of the module structuring decisions without consulting the designer. Table 46 reports the results of the module structuring for the robot controller system.

C.1.2.3 Integrating Tasks And Modules

Once both tasks and modules are structured, the designer decides to integrate the two views. CODA makes these decisions without consulting the designer.

C.1.2.4 Defining Task Interfaces

Only the task interfaces remain to be defined. To complete the design, CODA allocates the external interfaces and the event flows among the various tasks within the design. Subsequently, CODA considers the data flows between pairs of tasks. For several data flows, CODA cannot reach a definite decision regarding the form of message passing to use. Since the designer is experienced, CODA seeks guidance. The designer is free in each case to decline to assist CODA. Had the designer been inexperienced then CODA would simply make default decisions to map the data flows to queued messages, as is also the case when an experienced designer declines to provide guidance. The following paragraphs discuss each case where CODA seeks advice from the designer to map a data flow to a message.

Table 46. Module Structuring Decisions for Robot Controller, Version 1

Module	Transformation/Data Store	Structuring Criterion
Command Handler	Resume Program Stop Program Process Program Ended End Program Start Program	State-Dependent, Function Driver Module
Program ID	Program ID Change Program	Data-Abstraction Module Update Operation of DAM
Sensor/Actuator Database	Sensor/Actuator Data Process Actuator Command Process Sensor Command	Data-Abstraction Module Update Operation of DAM Operation of DAM
Robot Controller	Robot Controller	State-Transition Module
Actuator	Actuators	Device-Interface Module
Sensor	Sensors	Device-Interface Module
Axis	Axis Controller	Device-Interface Module
Control Panel Input	Operator Input Panel	Device-Interface Module
Control Panel Output	Operator Output Panel	Device-Interface Module
Axis Manager	Generate Axis Command Receive Acknowledgement	Algorithm-Hiding Module Designer Allocated Function
Interpreter	Interpret Program Statement Process Motion Command	Algorithm-Hiding Module Designer Allocated Function
Robot Program	Robot Program	Data-Abstraction Module

The first data flow in question, Axis Block, goes from the Axis Manager to the Axis Controller. CODA asks the designer whether the sender of an Axis Block must synchronize with the receiver. The designer indicates that synchronization is necessary. With this additional information, CODA allocates Axis Block to a tightly-coupled message. In addition, CODA infers that Axis Acknowledgement, flowing in the reverse direction between the same pair of tasks, can also be allocated to a tightly-coupled message. The second set of data flows in question go from the Robot Command Processor task to the Control Panel Output Handler task. These data flows represent requests to light and extinguish various lamps on the control panel. In response to queries from CODA, the designer indicates that no synchronization is needed between the sender and receiver for these data flows. With this information, CODA allocates these data flows to a queued message.

The two situations described in the previous paragraph, indicate the dilemma faced by CODA when mapping data flows between tasks. Both the first data flow and the second set of data flows arrive at a device-output task. In one case synchronization is necessary, in the other case synchronization is not necessary. Each such decision requires application-specific knowledge. CODA possesses only general, design knowledge, and, therefore, must consult an experienced designer in the hope that the designer possesses the missing, application-specific knowledge. Absent such assistance, CODA maps the questionable data flows to queued messages.

Two additional variations of this same dilemma occur in the robot controller case study. The Robot Command Processor task sends a data flow, Start Program Command, to the Interpreter task. CODA cannot infer the synchronization needs for this data flow. The designer is consulted, indicating that synchronization is needed. With this knowledge, CODA allocates the data flow to a tightly-coupled message between the two tasks. In the second instance, the task Interpreter sends a data flow, Motion Block, to the Axis Manager task. Lacking sufficient knowledge to allocate this data flow to a message, CODA consults with the designer who indicates that synchronization is not needed. CODA then allocates the data flow to a queued message between the two tasks. CODA makes an additional inference that the data flow, Motion Acknowledgements, going in the reverse direction, can be allocated to the same type of message.

CODA is faced with one additional difficulty. The data/control flow diagram is constructed in a form such that one transformation, Generate Axis Command, sends a data flow, Axis Block, to a device object, Control Axis, and that device object sends another data flow, Axis Acknowledgement, to a different transformation, Receive Acknowledgement. Logically, one of those data flows, Receive Acknowledgement, is sent as a response to the other, Axis Block; however, CODA cannot infer that fact because during classification both data flows appeared to be independent. CODA assumes, though, that when a device input/output object receives a tightly-coupled message from a task and also sends a tightly-coupled message to the same task, the sent message is a reply to the received message. When the designer is not experienced,

CODA makes such a decision automatically. When the designer is experienced, CODA consults with the designer. The experienced designer is free to guide CODA or to decline to give any guidance. If the designer provides guidance, then CODA acts on it. When the designer provides no guidance, CODA takes the same, default action used with an inexperienced designer. In the case study, the designer provides guidance that is consistent with the default decision and CODA maps the message flowing from the device input/output task as an answer to the message received by the task.

From this point, CODA finishes the task-interface definition by considering whether priority messages might be needed and then by allocating queuing mechanisms, as required, for tasks. The designer is then invited to review the new task-interface elements and to rename them.

C.1.2.5 The Completed Design

The design generated by CODA, with guidance from an experienced designer, is shown in Figure 55. This design is almost identical to the solution given by Gomaa in his case study. The only structural difference appears with regard to module structuring. Gomaa uses application-specific knowledge to merge two modules, Command Handler and Program ID, from Figure 55, into a single module. CODA does not contain the required knowledge, nor does CODA contain a rule for recognizing that an experienced designer should be consulted on the question of merging these modules. Gomaa's solution appears superior on this point; however, the solution generated by CODA is also reasonable. A minor difference appears with regard to the number of operations allocated

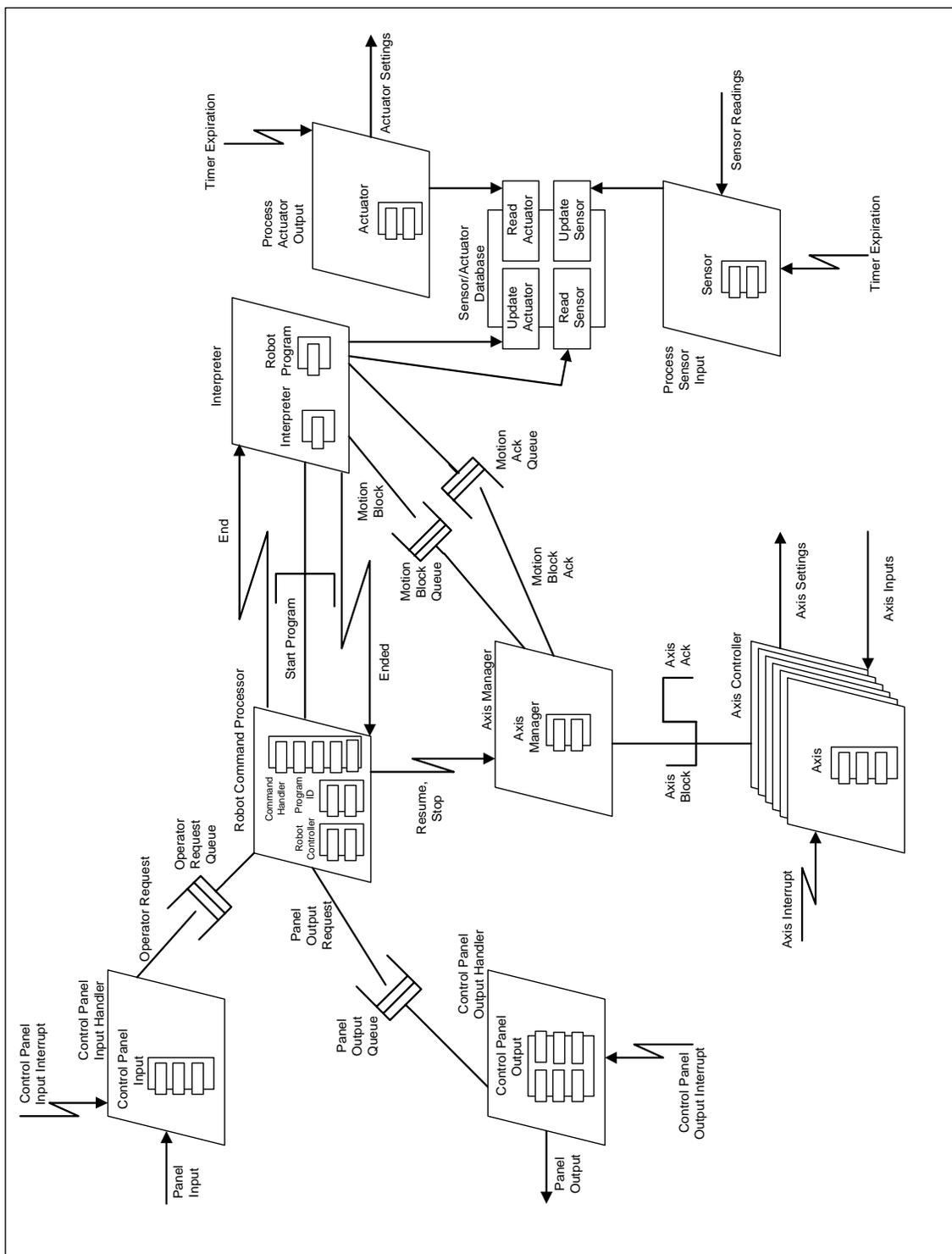


Figure 55. Generated Design for Robot Controller - Default Target Environment

to each module. CODA creates a larger number of operations for several of the modules. This results from the strategy CODA uses to map specification elements to operations. A human designer is expected to optimize these results, as desired.

C.1.3 A Design for Target Environments without Message Queues

To demonstrate another of CODA's capabilities, the designer decides to reuse a partially completed design. In this example, the designer reuses the design created previously from the robot controller specification. At each stage in the design process, the designer saved the state of the design. Now, in order to move the robot controller system to a target environment that does not support message queuing, the designer first copies a partially-completed, old design into a workspace for the new design and then continues the new design from the preexisting state of the old design. For the current prototype, the process of renaming and copying old designs is handled outside of CODA by using file renaming and copying services from the hosting operating system. Prior to reusing this design, the designer loads a new target environment description, NOQUEUE, for a system that provides no message queuing services. When the designer next asks CODA to generate a design, CODA detects that a design already exists in the workspace, loads the design, and reports the state of the design to the designer. In this case study, the old design already has an integrated task and module structure. This is exactly what the designer needs because only the task interfaces must be changed to correspond to the new target environment description. CODA then gives the designer an option to continue the existing design or to start a new design. In this example, the designer continues the

existing design and CODA generates new task interfaces that correspond to requirements of the new target environment. The completed design, built by reusing much of a preexisting design, is shown in Figure 56.

The design is identical to that generated previously, see Figure 55, except that each of the four message queues is now embedded inside a queue-control task. Access to these queue-control tasks is made via tightly-coupled messages. A Send message submits an element to be queued. A Receive Request asks for the next queued message. A Receive Reply returns the next queued message.

C.2 Robot Controller System, Version 2

The data/control flow diagram for the robot controller system, as provided by Gomaa, leads to several consultations between CODA and the designer because multiple, data transformations, Generate Axis Command and Receive Acknowledgement, interact with the Axis Controller device object. Recall, for example, that CODA consulted with the designer concerning the allocation of Receive Acknowledgement to a task. Similar consultation was needed during module structuring. Further, when task interfaces were defined, CODA consulted the designer about both the type of and the relationship between a pair of data flows, Axis Block and Axis Acknowledgement, exchanged between the Axis Manager and Axis Controller tasks. These consultations could be avoided by modifying the data/control flow diagram slightly so that a single transformation subsumes both Generate Axis Command and Receive Acknowledge. Such modifications bring the data/control flow diagram more closely in alignment with

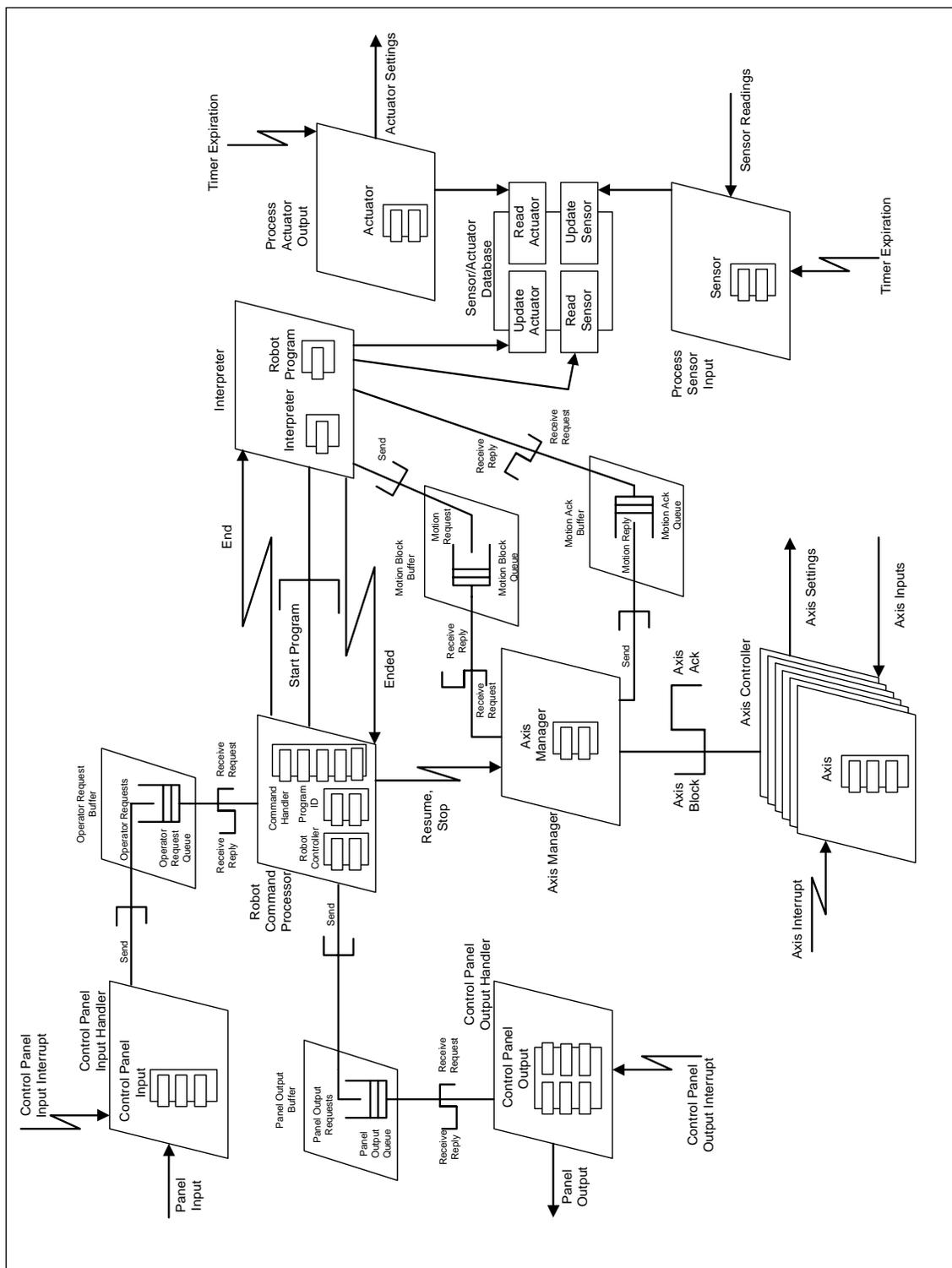


Figure 56. Generated Design for Robot Controller - Target Environment Provides No Message Queues

the COBRA restrictions on the use of RTSA notation. Figure 57 shows such a modification to the original data/control flow diagram.

Notice that the data flow Axis Acknowledgement, previously classified as a Stimulus, is now classified as a Response. CODA makes this classification without consulting the designer. As a result, CODA later understands, also without consulting the designer, how to map this interface to a tightly-coupled message with reply. Since the Receive Acknowledgement transformation is no longer present, CODA does not need to consult the designer about how to allocate that transformation. A designer used CODA to generate a design, shown in Figure 58, from the modified data/control flow diagram. The design is generated for the default target environment. Figure 58 reveals that the design is identical to that obtained from the original version of the data/control flow diagram. This comparison demonstrates that certain data/control flow diagrams, those constructed in accordance with the COBRA restrictions on the use of RTSA notation, can be analyzed more automatically by CODA than can other diagrams, that is, those constructed using unrestricted RTSA notation. In general, any synchronous function, where that function links with two or more transformations, might be allocated to the same task as one or more connected transformations. CODA cannot determine the best allocation of a synchronous function among multiple, connected transformations. A designer might be able to determine the best allocation. For this reason, CODA consults a designer whenever one of these situations arises. If the designer cannot help, then CODA makes a default decision. For task allocation, the default decision creates a separate task for the

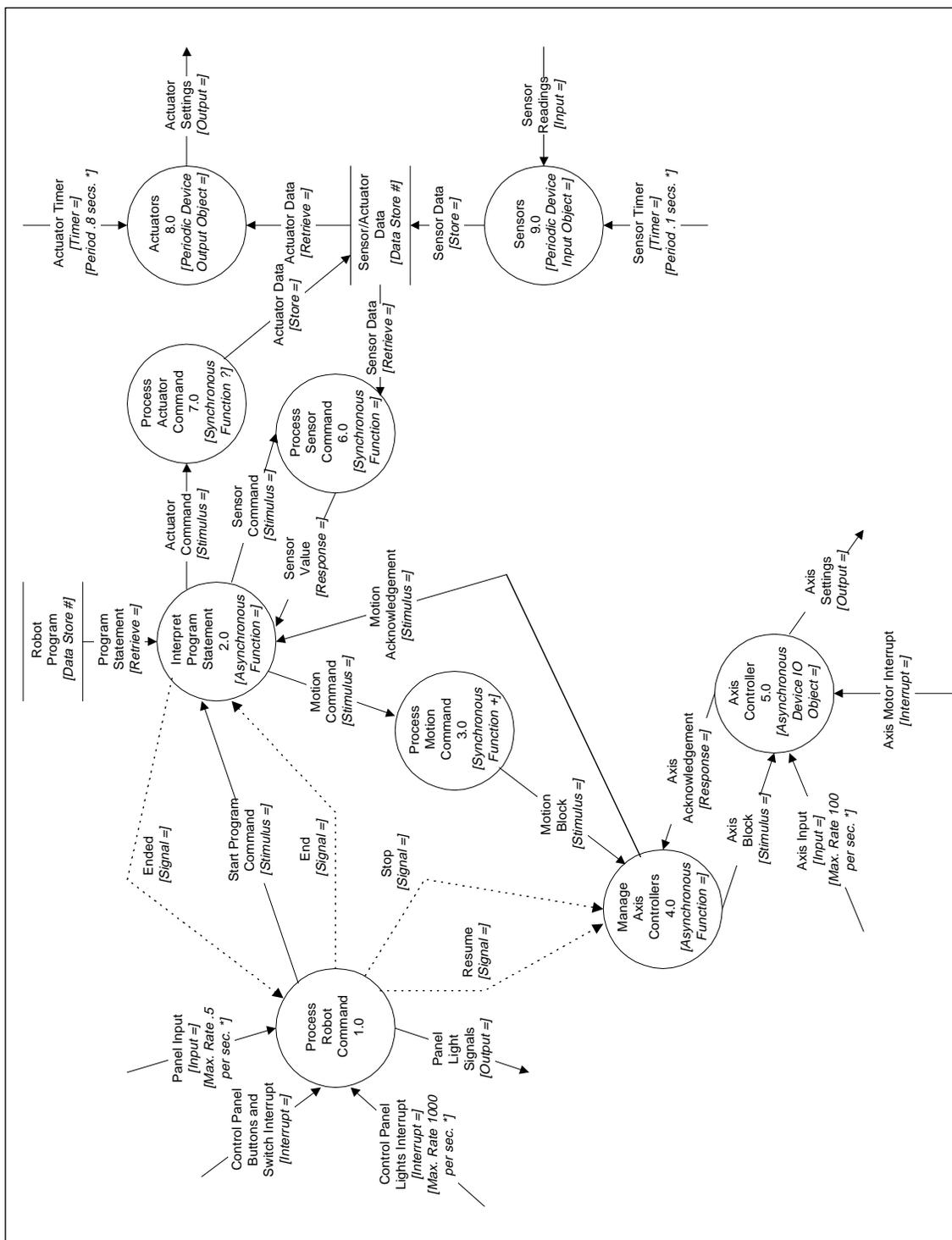


Figure 57. A Modified Data/Control Flow Diagram for the Robot Controller

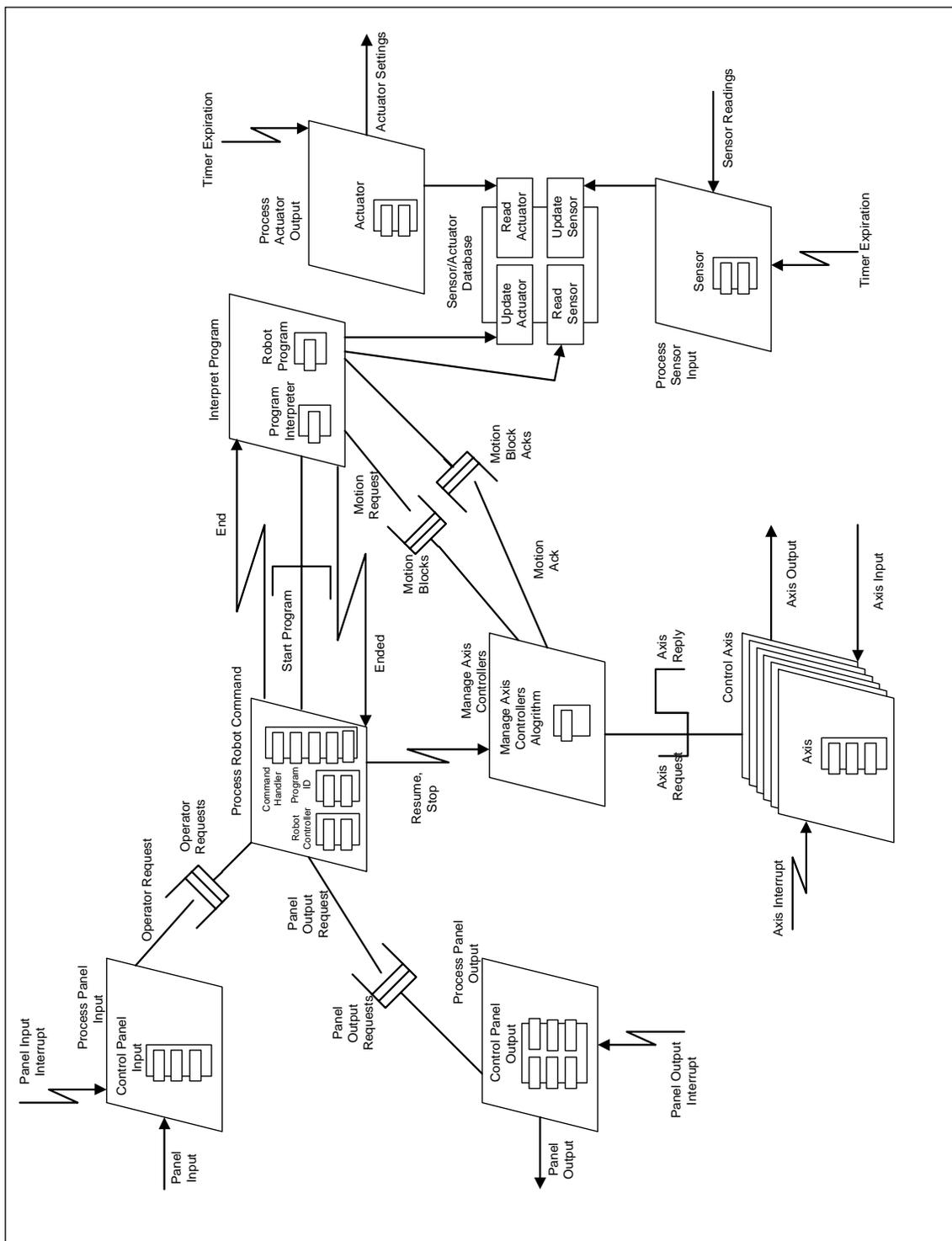


Figure 58. Generated Design for Robot Controller System, Version 2 - Default Target Environment

synchronous function in question. Similarly, for module allocation, the default decision allocates a separate module for the synchronous function in doubt. These decisions, while not incorrect, seldom lead to an efficient design.