

Appendix B. Automobile Cruise Control and Monitoring System Case Study

This appendix presents an application of the proof-of-concept prototype, CODA, described in Chapter 10, to an automobile cruise control and monitoring system. The specification for this system consists of a set of data/control flow diagrams, arranged hierarchically, two state-transition diagrams, and a textual description. The specification is taken from Gomaa.¹ [Gomaa93, Chapter 22] Figure 34 shows the context diagram for the problem. The context diagram is annotated with information inferred, or elicited from the designer, as a result of applying the prototype to analyze the specification. The annotations are shown on the diagram, and on all subsequent data/control flow diagrams, enclosed within square brackets and set off in italicized print. Symbols, defined in Table 31, are used with each annotation to indicate the source of the information.

The context diagram depicted in Figure 34 differs from Gomaa's context diagram in only two ways. First, events arriving from external devices are shown in Figure 34 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. These events must be added to the data/control flow diagram to allow CODA to make proper inferences about the terminators. Second, some of the elements in the diagram are named

¹The automobile cruise control and monitoring system is a well-known, real-time problem that is often cited in the literature. The interested reader might wish to consult other treatments of this problem. [Bollinger88, Brackett87, Caromel93, Gomaa94, Hatley87, Jones89, Jones90, Jones94, Mellor86, Sanden94, Shaw95, Smith88]

differently from the names assigned by Goma. This renaming allows each element in the specification to have a unique name.

Table 31. Symbols Used to Annotate Data/Control Flow Diagrams

Symbol	Meaning
#	This classification is directly representable in the Specification Meta-Model.
@	CODA elicited this classification from the user.
=	CODA made this classification.
?	CODA tentatively made a classification, but the user was asked to confirm or override that classification.
+	CODA elicited additional information from the user and then made this classification based on that additional information.
*	CODA elicited this information from the user.

Data/control flow diagrams for most large systems are arranged hierarchically to help human beings better comprehend the specification. CODA, however, works from a flattened hierarchy of data/control flow diagrams. The hierarchical form of the data/control flow diagrams is retained in the following discussion for clarity of exposition. Each annotated specification element that appears on a data/control flow diagram represents a specification element that cannot be further decomposed. For example, each terminator on the context diagram cannot be decomposed, and thus the

terminators exist as part of the flattened hierarchy seen by CODA. The same is true for the directed arcs flowing to and from the terminators. Contrast these with the data transformation, Automobile Cruise Control and Monitoring, in the context diagram. This data transformation is not annotated because it can be decomposed on additional diagrams.

B.1 Analyzing the Specification

The data/control flow diagram for this case study consists of 58 nodes (33 transformations, 12 terminators, and 13 data stores) and 112 arcs (69 data flows and 43 event/control flows). The designer invokes CODA for assistance in analyzing the data/control flow diagram and then in generating a concurrent design.

B.1.1 Classifying the Specification

Classifying the specification requires a dialog between CODA and the designer; however, CODA can make most classification decisions without consulting the designer. In this case study, only two consultations are necessary. First, the designer is asked if all terminators in the specification are devices. The designer indicates that all terminators are devices. Second, during the latter stages of classification, CODA discovers six data transformations that appear to be synchronous functions. Knowing the designer to be experienced, CODA presents each of these tentative classifications to the designer for confirmation. The designer confirms that the data transformations, Initialize MPH, Determine Speed, Initialize MPG, Initialize Oil Filter, Initialize Air Filter, and Initialize Major Service, do represent synchronous functions.

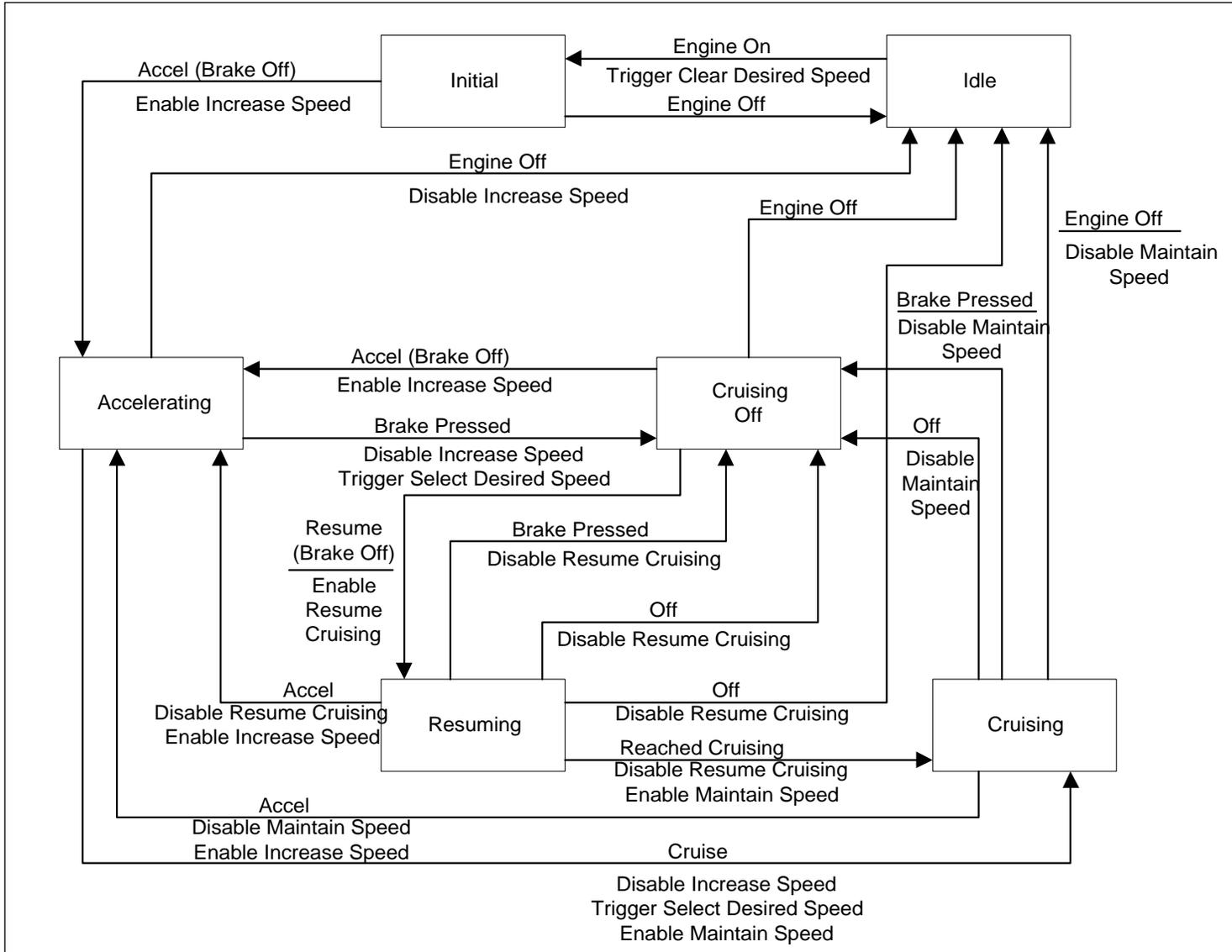
B.1.2 Eliciting Additional Information

Since CODA determined new semantic identities for specification elements in the data/control flow diagram, CODA checks each specification element to determine if additional information must be supplied by the designer. In this case study, sixteen event flows represent timers. CODA forces the designer to provide a positive period for each timer. The designer takes the periods from the textual description for the automobile cruise control and monitoring system. CODA also finds two system inputs from asynchronous devices, the cruise control lever and the shaft. CODA requires the designer to provide a value for a maximum rate at which these inputs are expected to arrive.

CODA discovers that no exclusion groups exist in the specification, and so, offers the designer a chance to specify exclusion groups. The designer, after reading the textual specification and manually analyzing the state-transition diagram, shown in Figure 35, associated with Cruise Control, determines that Maintain Speed is enabled only in state Cruising, Increase Speed is enabled only in state Accelerating, and Resume Cruising is enabled only in state Resuming. The designer concludes that none of the controlled data transformations execute simultaneously; thus, the designer specifies an exclusion group that includes all three of the data transformations. This specification addendum helps CODA identify instances where candidate tasks can be combined based upon mutual exclusion.

Next, CODA determines that no aggregation groups nor locked-state events exist within the specification and then offers the designer an opportunity to add these addenda.

Figure 35. State-Transition Diagram for Cruise Control



In this case study, the designer provides no aggregation groups and no locked-state events. Finally, CODA asks the designer if the cardinality of any nodes should be altered. The designer decides not to change any cardinalities.

B.1.3 Checking Classifications and Axioms

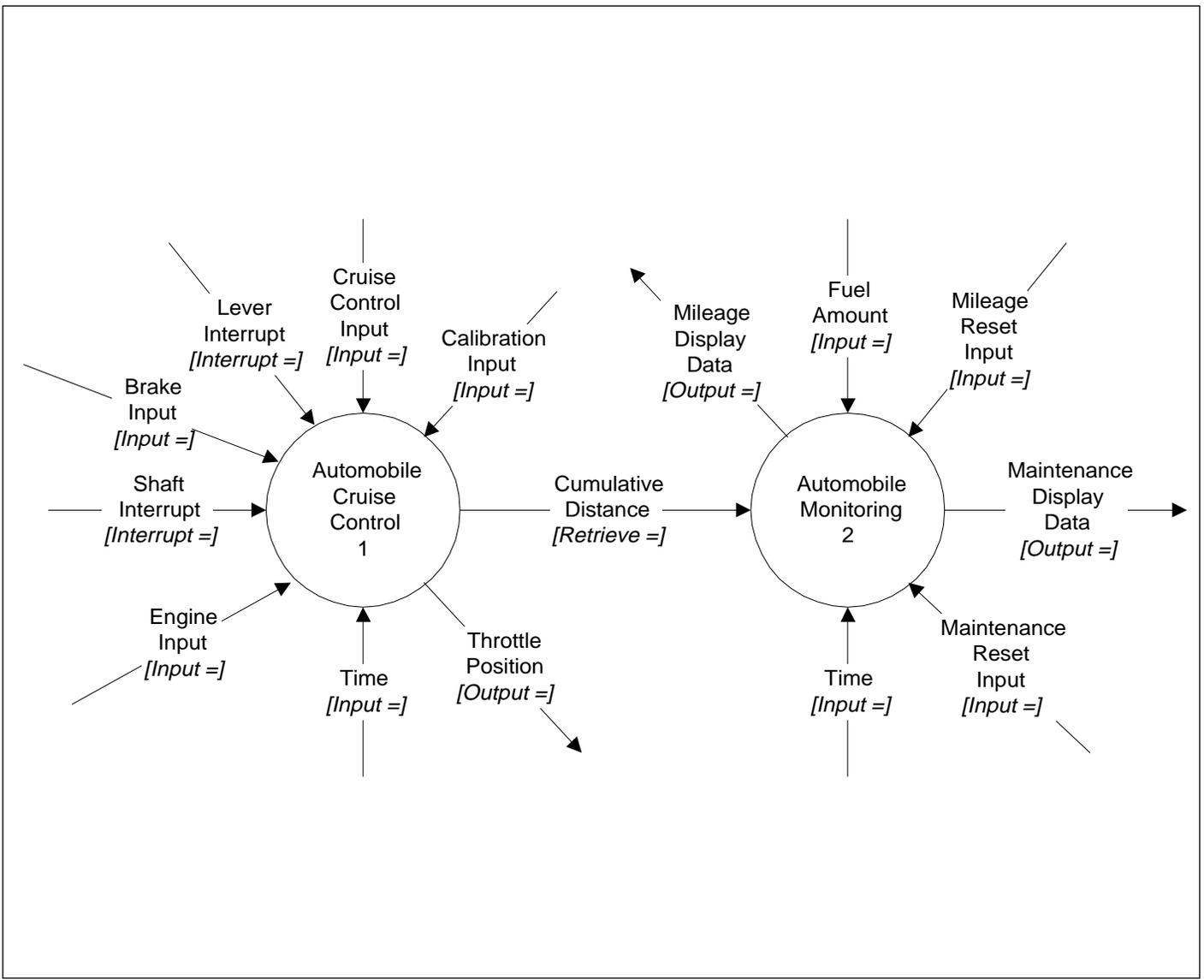
The designer next asks that CODA check the classification and axioms for each specification element. CODA finds that the specification is classified completely and that all axioms are satisfied. To better understand specification analysis for the automobile cruise control and monitoring system, the annotated data/control flow diagrams are presented in Figures 36 through 44.

B.1.4 Annotated Data/Control Flow Diagrams

Figure 36 shows that the application divides into two subsystems: an automobile cruise control subsystem and an automobile monitoring subsystem. (Refer back to Table 31 to review the symbols used to annotate the data/control flow diagrams.) The directed arcs flowing between the system and its terminators are allocated among these two subsystems. The interface between the subsystems consists of a data flow, Cumulative Distance, from the cruise control subsystem to the monitoring subsystem. CODA classifies Cumulative Distance as a Retrieve, indicating that this data flow originates at a data store within the cruise control subsystem. Since neither data transformation representing a subsystem is annotated, additional decomposition is required.

Figure 37 shows the decomposition of the automobile cruise control subsystem into three parts: Automobile Control (1.1), Distance and Speed Measurement (1.2), and

Figure 36. Subsystem Decomposition of Automobile Cruise Control and Monitoring



Calibration (1.3). At this level of decomposition, system inputs and outputs are allocated more finely; for example, the Automobile Control data transformation receives four inputs, Brake Input, Engine Input, Cruise Control Input, and Lever Interrupt, and generates one output, Throttle Position. This level of decomposition also reveals that the Cumulative Distance data flow originates within the Distance and Speed Measurement data transformation. Three new data flows also appear at this level of decomposition. CODA classifies each of these new data flows, Current Speed, Calibration Constant, and Shaft Rotation Count, as a Retrieve. This indicates that each of these data flows originates at a data store. None of the data transformations shown in Figure 37 is annotated, so each can be decomposed further.

Figure 38 decomposes Automobile Control into five data transformations. CODA classifies Brake and Engine as Periodic Device Input Objects, and classifies Throttle as a Periodic Device Output Object. Each of these data transformations receives an event flow, Brake Sensor Timer, Engine Sensor Timer, and Throttle Output Timer, respectively, that CODA classifies as a Timer. The period for each timer, 1/10 second in each case, is elicited from the designer. CODA classifies Cruise Control Lever as an Asynchronous Device Input Object. Cruise Control Lever receives the event flow, Lever Interrupt, from the terminator, Four-Position Lever. A data flow, Cruise Control Input, arrives at a maximum rate, elicited from the designer. Figure 38 reveals five event flows and one data flow that do not appear at higher levels in the decomposition hierarchy. Speed Control receives two event flows, Brake Pressed and Brake Released, from Brake

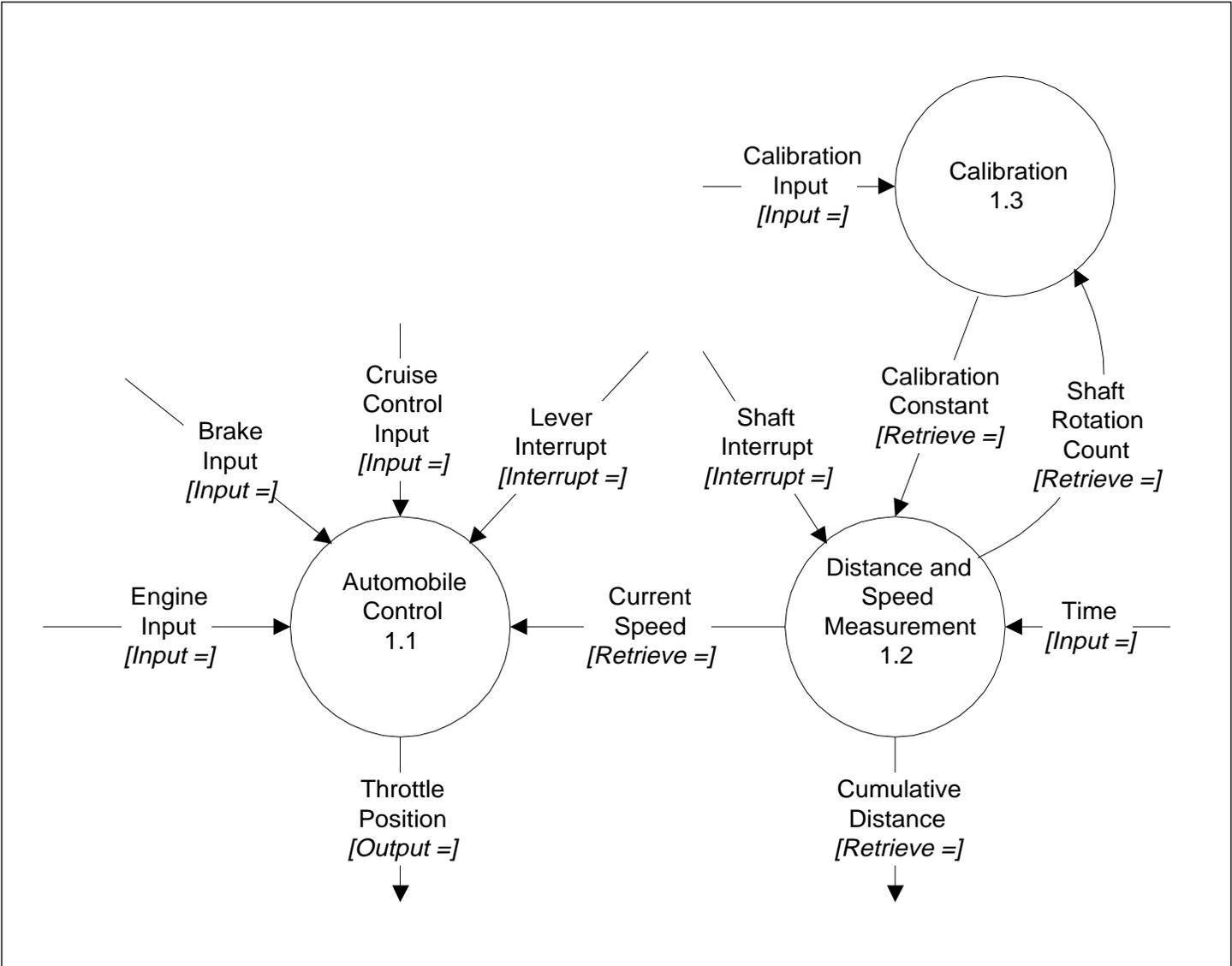


Figure 37. Decomposition of the Automobile Cruise Control Subsystem

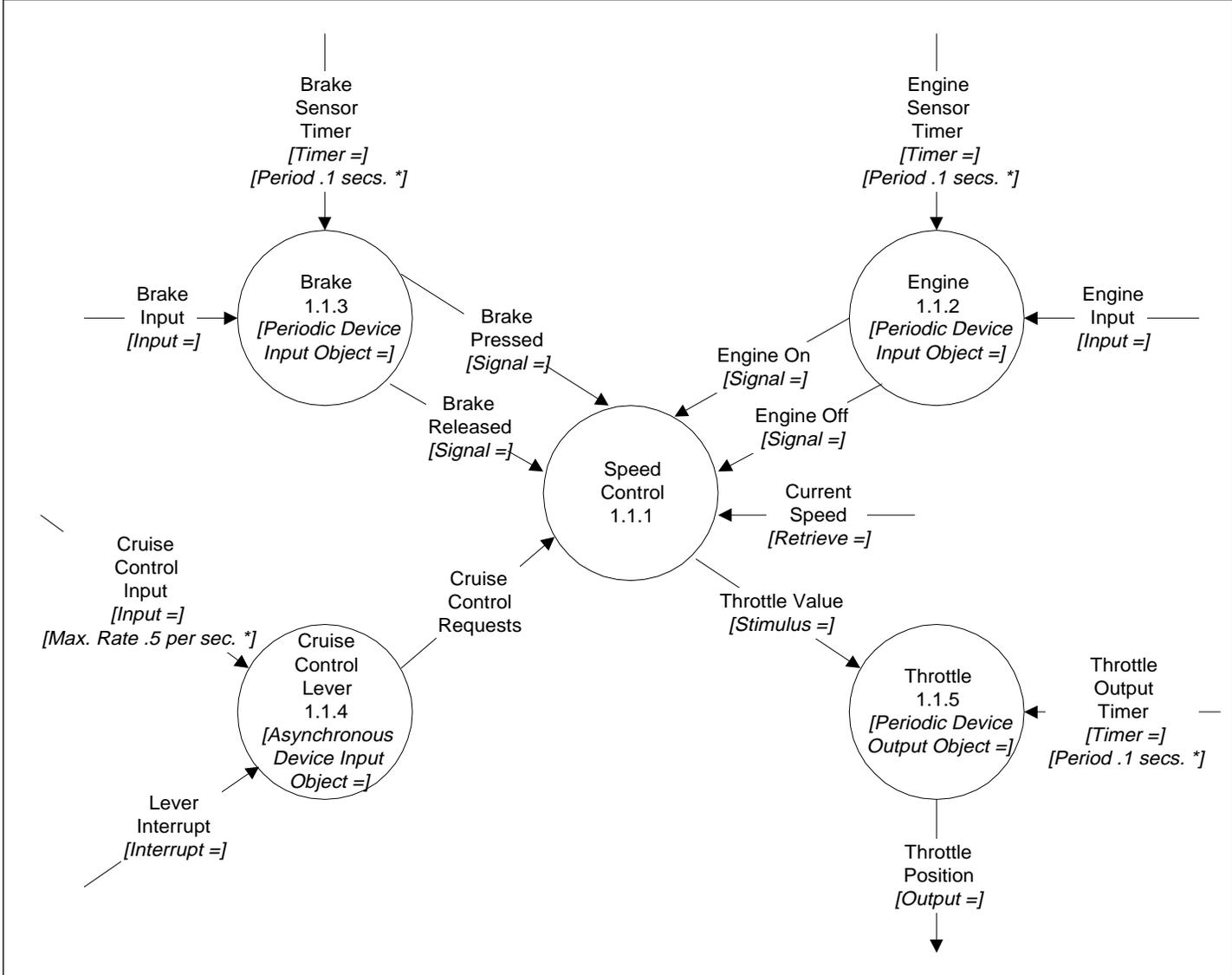


Figure 38. Decomposition of Automobile Control

and two, Engine On and Engine Off, from Engine. CODA classifies each of these event flows as a Signal. The fifth new event flow, Cruise Control Requests, originating from Cruise Control Lever, is not annotated in Figure 38 because the event flow is decomposed on a subsequent diagram. The data flow, Throttle Value, received by Throttle is classified by CODA as a Stimulus. One data transformation, Speed Control, is not annotated because it can be decomposed further.

Speed Control is decomposed, as shown in Figure 39, into one control transformation, five data transformations, and a data store. In addition, the event flow named Cruise Control Requests, is decomposed into four individual event flows, Accel, Cruise, Resume, and Off. CODA classifies each of these event flows as a Signal. CODA also classifies the control transformation, Cruise Control, as a Control Object. Two data transformations, Select Desired Speed and Clear Desired Speed, are classified by CODA as Triggered Synchronous Functions. CODA classifies each of the three remaining data transformations, Maintain Speed, Resume Cruising, and Increase Speed, as an Enabled Periodic Function. The data store, Desired Speed, can be represented directly with the specification meta-model; thus, no classification is required. Twelve new event flows are revealed at this level of decomposition. CODA classifies two event flows as Triggers, three event flows as Enables, three event flows as Disables, one event flow, Reached Cruising, as a Signal, and three event flows, Speed Timer, Resume Timer, and Increase Timer, as Timers. CODA elicits the period associated with each Timer. Four new data flows are revealed at this level of decomposition. CODA classifies two of these as Stores

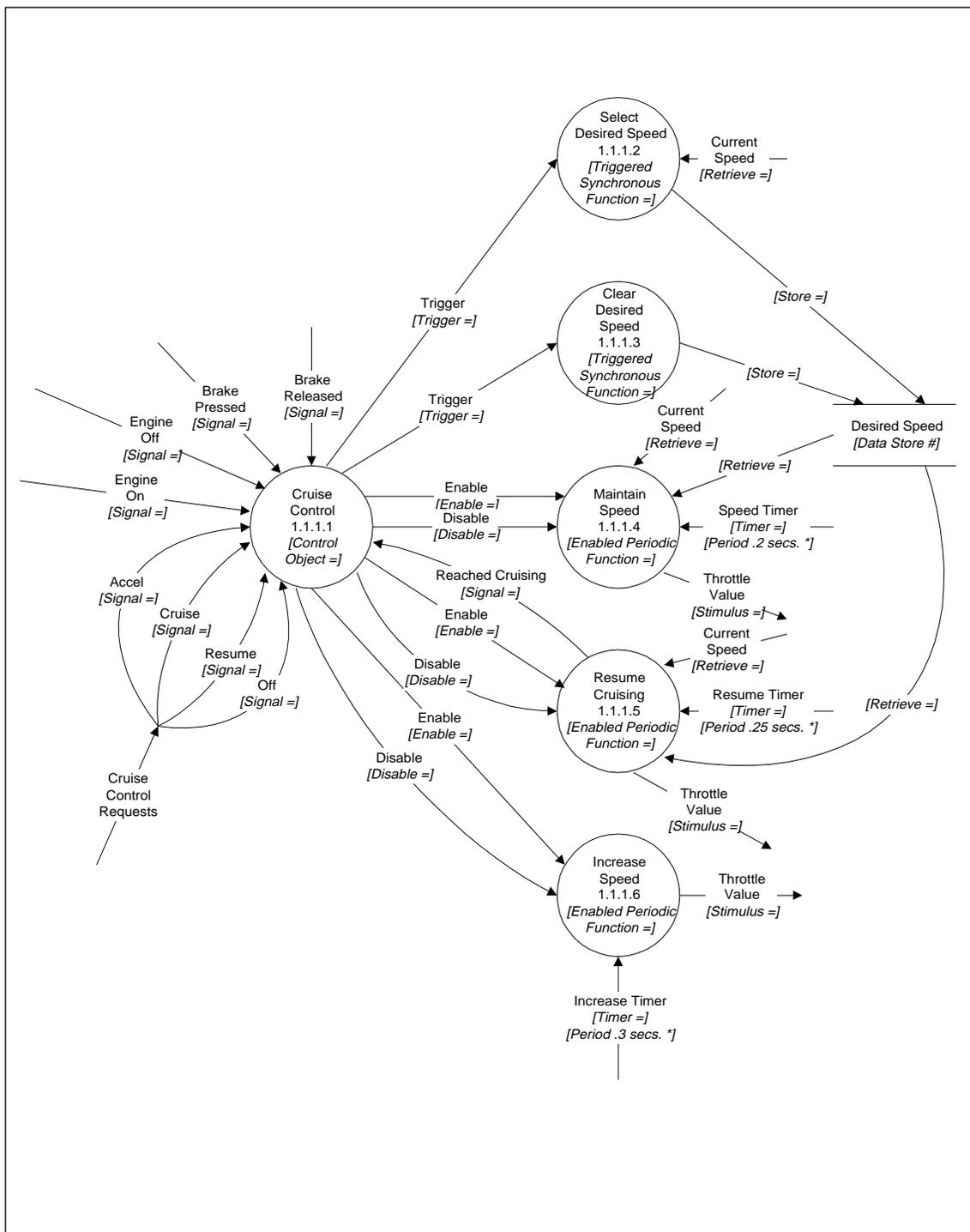


Figure 39. Decomposition of Speed Control

to a data store, Desired Speed, and the other two as Retrieves from Desired Speed.

Recall from Figure 37 that the data transformation named Distance and Speed Measurement can be further decomposed. This decomposition, shown in Figure 40, includes three data transformations, Shaft, Determine Distance, and Determine Speed, and five data stores, Current Speed, Shaft Rotation Count, Last Distance, Cumulative Distance, and Last Time. CODA classifies Shaft as an Asynchronous Device Input Object and elicits the maximum rate of arrival for the Shaft Interrupt event flow. CODA classifies the new event flow, Distance Timer, as a Timer, elicits the period of 1/10 second for this Timer, and classifies Determine Distance as a Periodic Function. CODA classifies as an Update each of the two, two-way arcs that connect a data transformation to a data store. Each of the three directed arcs flowing to a data store is classified as a Store, while each directed arc flowing from a data store is classified as a Retrieve. The newly revealed data flow, Incremental Distance, is classified by CODA as a Stimulus. The remaining data transformation, Determine Speed, is classified tentatively as a Synchronous Function and the designer is asked to confirm or override that classification. The designer confirms the classification. The data flow named Time Request is classified by CODA as a Stimulus and the data flow Current Time is classified as a Response.

Another data transformation, Calibration, from Figure 37 can be decomposed further, as shown in Figure 41. In the same way as described for earlier figures, CODA makes the appropriate classifications for the specification elements on this diagram. This completes the decomposition of the Automobile Cruise Control subsystem.

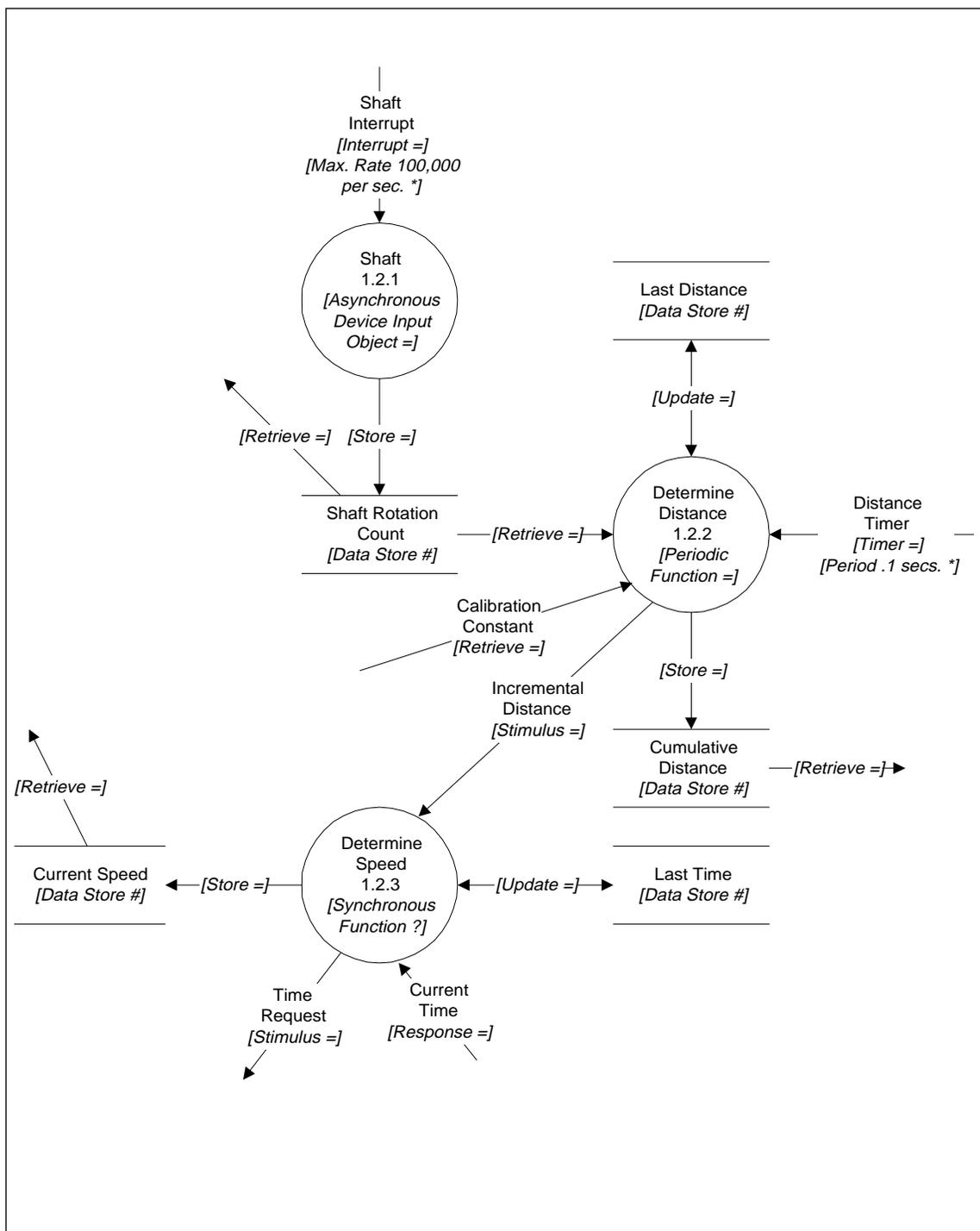


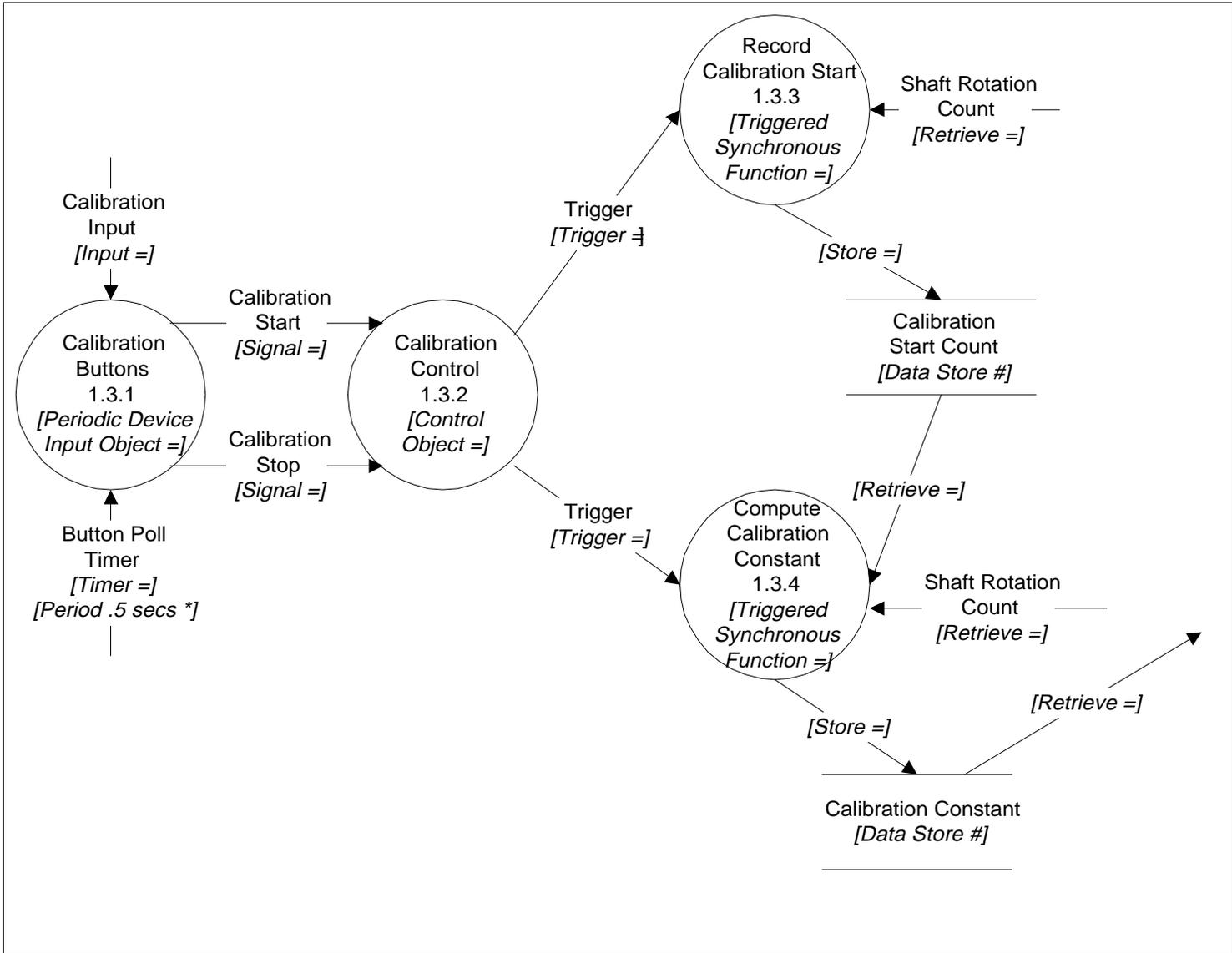
Figure 40. Decomposition of Distance and Speed Measurement

The Automobile Monitoring subsystem is decomposed in a similar fashion, beginning with Figure 42. The subsystem is divided into two components, Average Mileage and Maintenance, each represented by a data transformation that can be further decomposed. The inputs and outputs for the subsystem are divided between these two data transformations and the data flow, Cumulative Distance, from the Automobile Cruise Control Subsystem, is provided to both data transformations. Two new event flows, Mileage Timer Events and Maintenance Timer Events, are revealed at this level of decomposition. Each of these event flows can be further decomposed.

The decomposition for Average Mileage is illustrated in Figure 43. CODA makes the following classifications for data transformations: Clock and Gas Tank are Passive Device Input Objects; Mileage Display is a Passive Device Output Object; Mileage Reset Buttons becomes a Periodic Device Input Object; Compute Average MPH and Compute Average MPG are Periodic Functions. The two remaining data transformations, Initialize MPH and Initialize MPG, are tentatively classified as Synchronous Functions and the designer is asked to confirm or override this classification. The designer confirms the classifications. The new event and data flows revealed in this decomposition are classified by CODA. CODA also elicits periods from the designer for each event flow identified as a Timer.

The final portion of the data/control flow diagram, shown in Figure 44, decomposes the Maintenance data transformation from Figure 42. CODA assigns the following classifications to data transformations in Figure 44: Maintenance Reset

Figure 41. Decomposition of Calibration



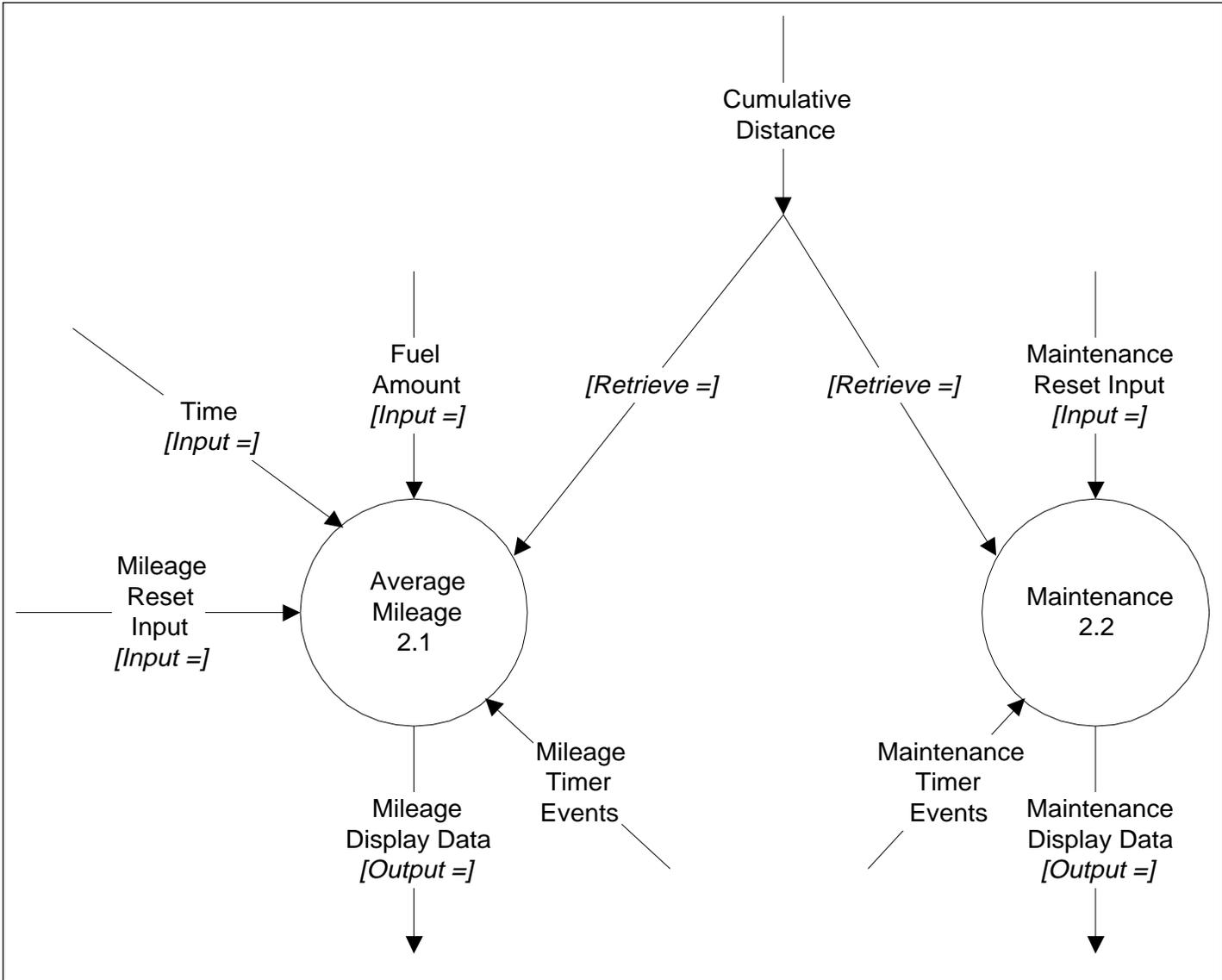


Figure 42. Decomposition of the Automobile Monitoring Subsystem

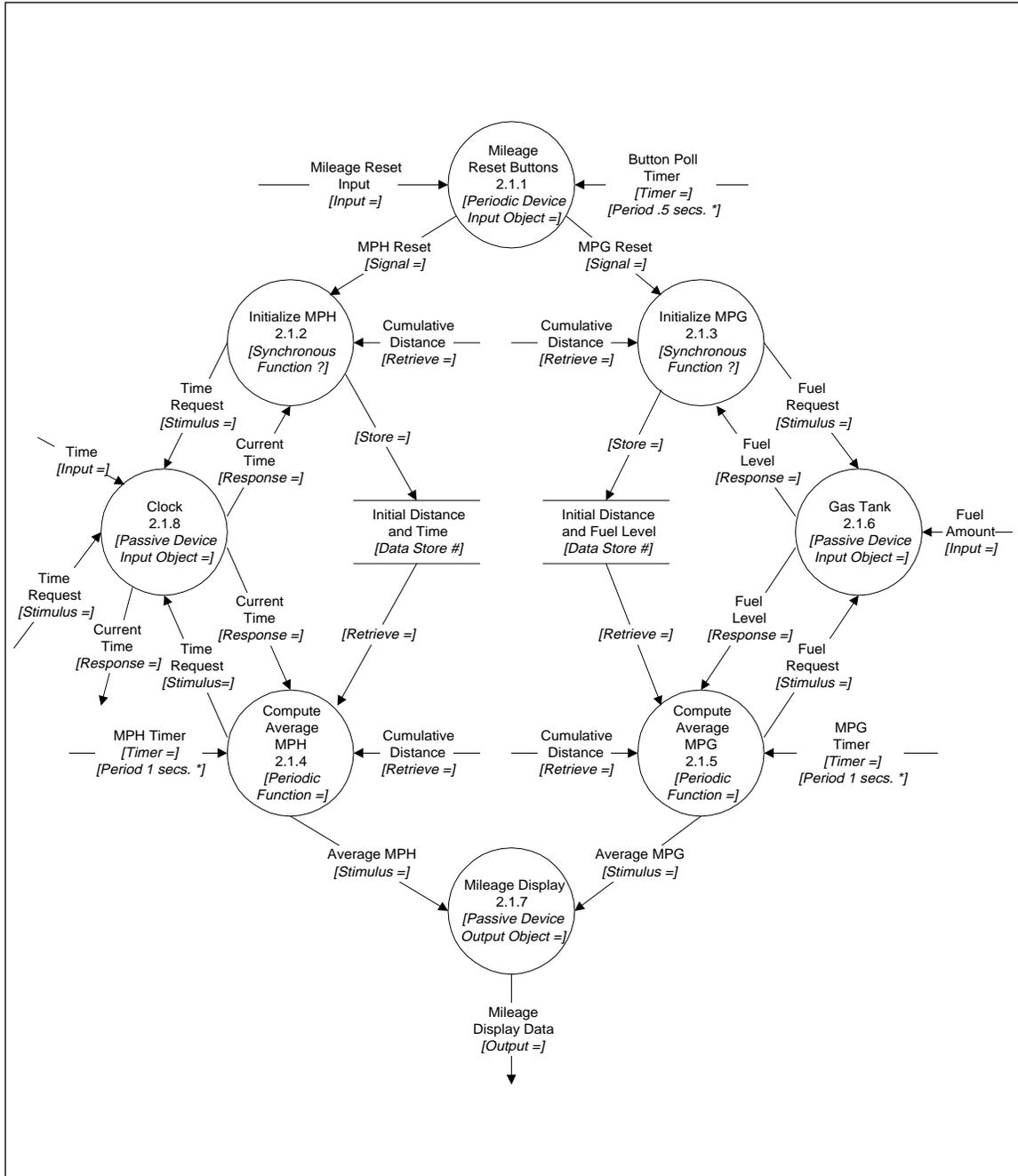


Figure 43. Decomposition of Average Mileage

Buttons becomes a Periodic Device Input Object; Maintenance Display becomes a Passive Device Output Object; Check Oil Filter Maintenance, Check Air Filter Maintenance, and Check Major Service Maintenance each become a Periodic Function. CODA makes tentative classifications for the following Synchronous Functions: Initialize Oil Filter, Initialize Air Filter, and Initialize Major Service. The designer confirms these classifications. The newly revealed event and data flows are classified by CODA and periods are elicited for each event flow classified as a Timer.

B.2 Generating the Design

After analyzing the data/control flow diagram, the designer decides to generate a concurrent design, beginning with task structuring. First, the designer must load a target environment description. For this case study, the designer chooses a DEFAULT target environment description, including the following characteristics of note: a maximum of two inter-task signals, a task inversion threshold of eight, support for message queues, and no support for priority queues. The designer then asks CODA to structure tasks for the design.

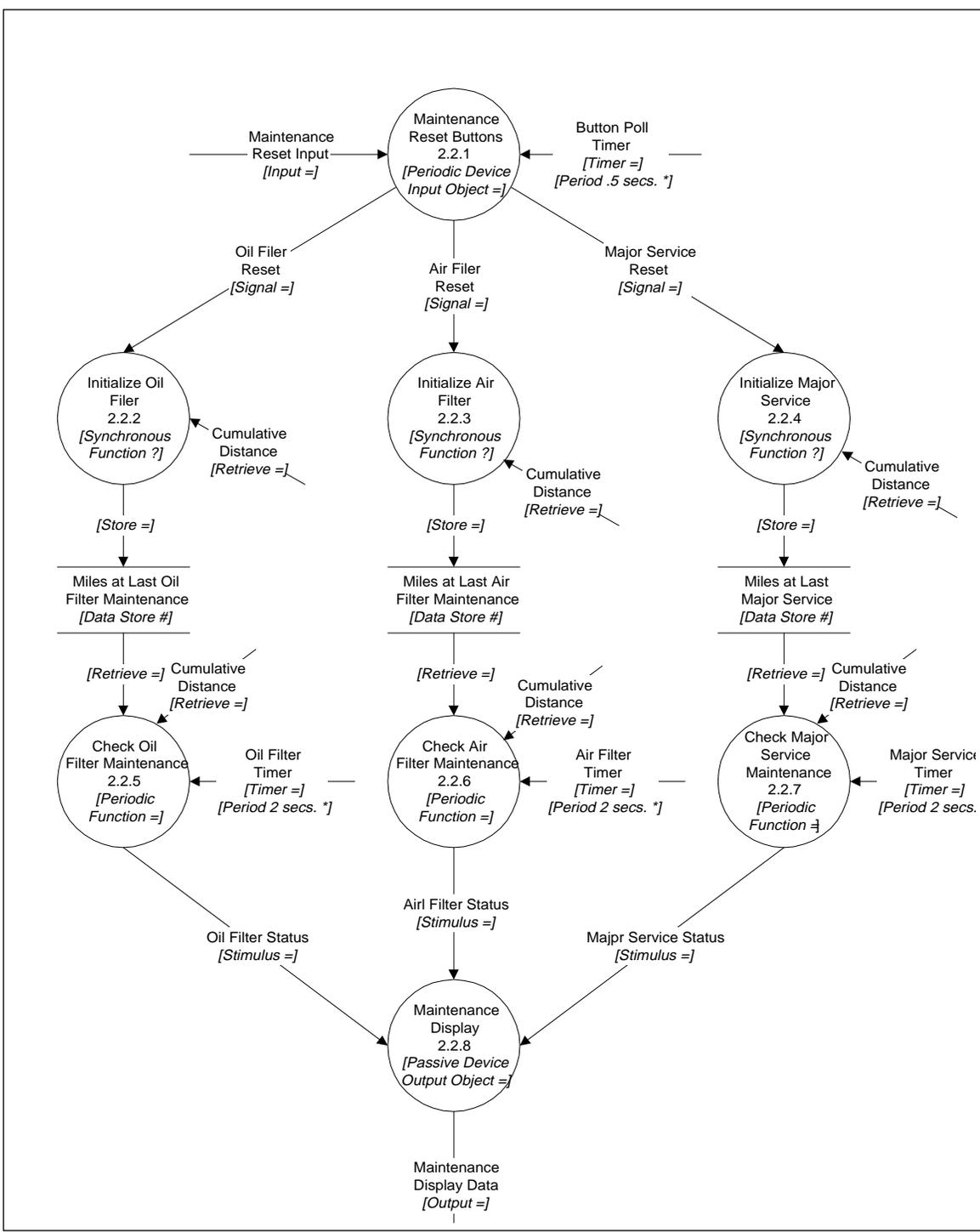


Figure 44. Decomposition of Maintenance

B.2.1 Structuring Tasks

For this case study, CODA makes most task structuring decisions without consulting the designer; however, since the designer is experienced, CODA does consult concerning one possible decision to merge two tasks. The following discussion reflects the decision-making processes used by CODA to structure tasks.

B.2.1.1 Identifying Candidate Tasks

CODA begins by allocating candidate tasks from each of three transformations, Increase Speed, Maintain Speed, and Resume Cruising, based on a CODARTS criterion for identifying controlled, periodic internal tasks. Next, CODA allocates a task from each of six transformations, Determine Distance, Compute Average MPH, Compute Average MPG, Check Oil Filter Maintenance, Check Air Filter Maintenance, and Check Major Service Maintenance, based on a CODARTS criterion for identifying periodic internal tasks. CODA allocates two more tasks from among internal transformations, Cruise Control and Control Calibration, based on the CODARTS criterion for identifying control tasks. CODA's remaining allocation of candidate tasks comes from device interface objects. CODA allocates a task from each of six device interface objects, Brake, Engine, Throttle, Mileage Reset Buttons, Maintenance Reset Buttons, and Calibration Buttons, based on the CODARTS criterion for identifying periodic input/output tasks and allocates a task from each of two device interface objects, Cruise Control Lever and Shaft, based on the CODARTS criterion for identifying asynchronous input/output tasks. Table 32 shows CODA's candidate tasks at the end of this decision-making process.

Table 32. Candidate Tasks Allocated by CODA

Candidate Task	Transformation	Structuring Criterion
Task 1	Increase Speed	Controlled Periodic Internal Task
Task 2	Maintain Speed	Controlled Periodic Internal Task
Task 3	Resume Speed	Controlled Periodic Internal Task
Task 4	Determine Distance	Periodic Internal Task
Task 5	Compute Average MPH	Periodic Internal Task
Task 6	Compute Average MPG	Periodic Internal Task
Task 7	Check Air Filter Maintenance	Periodic Internal Task
Task 8	Check Oil Filter Maintenance	Periodic Internal Task
Task 9	Check Major Service Maintenance	Periodic Internal Task
Task 10	Cruise Control	Control Task
Task 11	Control Calibration	Control Task
Task 12	Brake	Periodic Input/Output Task
Task 13	Engine	Periodic Input/Output Task
Task 14	Throttle	Periodic Input/Output Task
Task 15	Calibration Buttons	Periodic Input/Output Task
Task 16	Maintenance Reset Buttons	Periodic Input/Output Task
Task 17	Mileage Reset Buttons	Periodic Input/Output Task
Task 18	Cruise Control Lever	Asynchronous Input/Output Task
Task 19	Shaft	Asynchronous Input/Output Task

B.2.1.2 Allocating Remaining Transformations

Next, CODA examines the remaining, unallocated transformations, in an effort to allocate them to appropriate tasks based upon CODARTS criteria for sequential and control cohesion or upon guidance elicited from the designer. In this case, CODA needed no guidance from the designer. Table 33 shows the decisions made by CODA during this decision-making process.

Table 33. Additional Transformations Allocated to Tasks by CODA

Candidate Task	Transformations Added	Cohesion Criterion
Task 4	Determine Speed Clock	Sequential Cohesion Sequential Cohesion
Task 5	Clock Mileage Display	Sequential Cohesion Sequential Cohesion
Task 6	Gas Tank Mileage Display	Sequential Cohesion Sequential Cohesion
Task 7	Maintenance Display	Sequential Cohesion
Task 8	Maintenance Display	Sequential Cohesion
Task 9	Maintenance Display	Sequential Cohesion
Task 10	Select Desired Speed Clear Desired Speed	Control Cohesion Control Cohesion
Task 11	Record Calibration Start Compute Calibration Constant	Control Cohesion Control Cohesion
Task 16	Initialize Oil Filter Initialize Air Filter Initialize Major Service	Sequential Cohesion Sequential Cohesion Sequential Cohesion
Task 17	Initialize MPG Initialize MPG Clock Gas Tank	Sequential Cohesion Sequential Cohesion Sequential Cohesion Sequential Cohesion

B.2.1.3 Considering Task Mergers

During the next decision-making process, CODA examines the candidate tasks in an effort to combine tasks, where feasible. CODA makes most of the decisions, shown in Table 34, without consulting the designer.

Table 34. Tasks Combined by CODA

Tasks Combined	Cohesion Criterion
Task 1 Task 2 Task 3	Mutual Exclusion
Task 5 Task 6	Temporal and Functional Cohesion
Task 7 Task 8 Task 9	Temporal and Functional Cohesion
Task 11 Task 15	Sequential Cohesion
Task 12 Task 13	Temporal and Functional Cohesion
Task 16 Task 17	Temporal and Functional Cohesion

CODA combines three tasks (1-3) based on mutual exclusion because the constituent transformations, Increase Speed, Maintain Speed, and Resume Speed, reside in the same exclusion group. CODA combines Task 5 and Task 6, based on Compute Average MPH and Compute Average MPG, respectively, because both periodic internal

tasks operate with the same periodicity, one second. Similarly, CODA combines Task 7, Task 8, and Task 9, based on Check Air Filter Maintenance, Check Oil Filter Maintenance, and Check Major Service Maintenance, respectively, because these periodic internal tasks operate with identical periods, two seconds. CODA combines Task 11 and Task 15, based upon Control Calibration and Calibration Buttons, respectively, because Task 11 must always receive input from Task 15 before executing and Task 11 interacts with no other tasks. CODA combines the Brake and Engine tasks, Task 12 and Task 13, respectively, because these periodic input tasks operate with identical periods, 1/10 of a second. Similarly, CODA merges the Maintenance Reset Buttons and Mileage Reset Buttons tasks, Task 16 and Task 17, respectively, because these periodic input tasks operate with identical periods, 1/2 of a second.

Two tasks in the evolving design might be combined because their periods are multiples of one another, are within an order of magnitude, and are the closest such periods existing in the evolving design. CODA cannot make a decision because additional, application-specific, factors must be considered. After explaining this to the designer, CODA offers an opportunity to review the structure of the tasks involved in the decision. For this case study, the designer asks to review the tasks. CODA lists each task, along with the transformations allocated to each task; see Table 35. CODA then asks the designer whether to combine these tasks. In this case, the designer decides not to combine the tasks because they do not exhibit enough functional similarity.

Table 35. A Candidate Task Merger

Task	Transformations
Previously Combined Tasks 5 and 6 (1 second period)	Compute Average MPG Compute Average MPH Gas Tank Mileage Display Clock
Previously Combined Tasks 7, 8, and 9 (2 second period)	Check Major Service Maintenance Check Air Filter Maintenance Check Oil Filter Maintenance Maintenance Display

B.2.1.4 Completing Task Structuring

Next, CODA considers whether the design requires any resource monitor tasks. In this case study, no resource monitor tasks are needed. At this point, task structuring is essentially complete; however, since CODA generates names for each task it creates, the designer is offered an opportunity to review the task structure and to assign new names to any task. Table 36 gives the results of the task structuring for this case study, including the new names assigned to each task, the transformations allocated to each task, and the structuring criteria used to make the allocations.

B.2.2 Structuring Modules

After structuring tasks, the designer might continue building the design by either defining task interfaces or structuring modules. In this case study, the designer decides to structure modules first. CODA makes most of the module structuring decisions

Table 36. Summary of CODA's Task Structuring Decisions

Task	Transformations	Structuring Criterion
Determine Speed & Distance	Determine Speed Determine Distance Clock	Periodic Internal Task Sequential Cohesion
Control Cruising	Cruise Control Select Desired Speed Clear Desired Speed	Control Task Control Cohesion
Adjust Throttle	Throttle	Periodic Device I/O Task
Monitor Shaft Rotation	Shaft	Asynchronous Device I/O
Monitor Cruise Control Lever	Cruise Control Lever	Asynchronous Device I/O
Control Auto Speed	Increase Speed Maintain Speed Resume Cruising	Controlled Periodic Internal Tasks Mutual Exclusion
Perform Calibration	Control Calibration Calibration Buttons Compute Calibration Constant Record Calibration Start	Control Task Periodic Device I/O Task Sequential Cohesion Control Cohesion
Monitor Auto Sensors	Brake Engine	Periodic Device I/O Tasks Temporal & Functional Cohesion
Monitor Reset Buttons	Mileage Reset Buttons Maintenance Reset Buttons Initialize MPH Clock Initialize MPG Gas Tank Initialize Oil Filter Initialize Air Filter Initialize Major Service	Periodic Device I/O Tasks Temporal & Functional Cohesion Sequential Cohesion

Table 36. Summary of CODA's Task Structuring Decisions (cont.)

Task	Transformations	Structuring Criterion
Check Maintenance Need	Check Major Service Maintenance Check Air Filter Maintenance Check Oil Filter Maintenance Maintenance Display	Periodic Internal Tasks Temporal & Functional Cohesion Sequential Cohesion
Computer Average Mileage	Compute Average MPG Compute Average MPH Gas Tank Mileage Display Clock	Periodic Internal Tasks Temporal & Functional Cohesion Sequential Cohesion

without consulting the designer; however, since the designer is experienced, CODA consults in a few cases where an experienced designer might improve upon the decisions.

B.2.2.1 Identify Candidate Modules

CODA begins module structuring by considering which transformations and data stores should form the basis for information hiding modules. CODA finds three transformations to combine into a single State-Dependent Function-Driver Module, eleven data structures from which to allocate Data-Abstraction Modules, two transformations that form the basis for State-Transition Modules, and twelve transformations that lead to Device-Interface Modules. Table 37 reflects these decisions.

Table 37. Candidate Modules Allocated by CODA

Candidate Module	Transformation/Data Store	Structuring Criterion
FDM 1	Increase Speed Maintain Speed Resume Cruising	State-Dependent, Function-Driver Module
DAM 1	Desired Speed	Data-Abstraction Module
DAM 2	Shaft Rotation Count	Data-Abstraction Module
DAM 3	Current Speed	Data-Abstraction Module
DAM 4	Cumulative Distance	Data-Abstraction Module
DAM 5	Initial Distance & Time	Data-Abstraction Module
DAM 6	Initial Distance & Fuel Level	Data-Abstraction Module
DAM 7	Miles at Last Oil Filter Maintenance	Data-Abstraction Module
DAM 8	Miles at Last Air Filter Maintenance	Data-Abstraction Module
DAM 9	Miles at Last Major Service	Data-Abstraction Module
DAM 10	Calibration Start Count	Data-Abstraction Module
DAM 11	Calibration Constant	Data-Abstraction Module
STM 1	Cruise Control	State-Transition Module
STM 2	Calibration Control	State-Transition Module
DIM 1	Maintenance Display	Device-Interface Module
DIM 2	Mileage Display	Device-Interface Module
DIM 3	Throttle	Device-Interface Module
DIM 4	Clock	Device-Interface Module
DIM 5	Gas Tank	Device-Interface Module
DIM 6	Shaft	Device-Interface Module
DIM 7	Cruise Control Lever	Device-Interface Module
DIM 8	Calibration Buttons	Device-Interface Module
DIM 9	Maintenance Reset Buttons	Device-Interface Module
DIM 10	Mileage Reset Buttons	Device-Interface Module
DIM 11	Brake	Device-Interface Module
DIM 12	Engine	Device-Interface Module

B.2.2.2 Allocating Functions to DAMs

Next, CODA attempts to allocate any unallocated functions to the candidate Data-Abstraction Modules, or DAMs, identified in the previous decision-making process. For this case study, sixteen functions can be allocated using CODARTS criteria for structuring modules. Table 38 shows CODA's decisions.

Table 38. CODA's Decisions to Allocate Functions to DAMs

Candidate Module	Transformations Added	Structuring Criterion
DAM 1	Clear Desired Speed Select Desired Speed	DAM Update Operation DAM Update Operation
DAM 3	Determine Speed	DAM Update Operation
DAM 4	Determine Distance	DAM Update Operation
DAM 5	Initialize MPH Compute Average MPH	DAM Update Operation DAM Read Operation
DAM 6	Initialize MPG Compute Average MPG	DAM Update Operation DAM Read Operation
DAM 7	Initialize Oil Filter Check Oil Filter Maintenance	DAM Update Operation DAM Read Operation
DAM 8	Initialize Air Filter Check Air Filter Maintenance	DAM Update Operation DAM Read Operation
DAM 9	Initialize Major Service Check Major Service Maintenance	DAM Update Operation DAM Read Operation
DAM 10	Record Calibration Start	DAM Update Operation
DAM 11	Compute Calibration Constant	DAM Update Operation

B.2.2.3 Allocating Isolated Elements

Typically, CODA next considers allocating any transformations that remain unallocated to some module; however, in this case study, all transformations are allocated at this point, so CODA turns instead to examine the two data stores that remain unallocated. First, CODA consults the designer to ensure that the isolated data stores, Last Distance and Last Time, are used for local memory. With this additional information the prototype knows how to allocate the data stores. CODA allocates one data store, Last Distance, to DAM 4 and allocates the other, Last Time, to DAM 3. Absent an experienced designer, CODA would have reached the same decisions, by default, for these cases.

B.2.2.4 Considering Module Subsumption

Since the designer is experienced, CODA considers whether some of the data-abstraction modules are candidates to be combined. In this case study, the prototype finds that the module Cumulative Distance is the only module that reads from the module Calibration Constant; thus, Calibration Constant is a candidate to be subsumed by Cumulative Distance. The designer reviews the transformations and data stores allocated to each module and decides not to merge the two because the two modules are too dissimilar functionally. Next, the prototype finds that Calibration Constant is the only module that reads from the module Start Calibration and, so, offers these as candidates to

be combined. After reviewing the components of each module, the designer decides to combine them because the two exhibit a close functional relationship.

B.2.2.5 Completing Module Structuring

At this stage, the modules in the design are established and CODA considers the operations, and associated parameters, required by each module. After mapping transformations and arcs to module operations and parameters, CODA allows the designer to review the module and operation structure and to assign new names. Table 39 provides a summary of CODA's module structuring decisions for this case study.

Table 39. Summary of CODA's Module Structuring Decisions

Module	Transformation/Data Store	Structuring Criterion
Control Auto Speed	Increase Speed Maintain Speed Resume Cruising	State-Dependent, Function Driver Module
Desired Speed	Desired Speed Clear Desired Speed Select Desired Speed	Data-abstraction Module DAM Update Operation
Shaft Rotation Count	Shaft Rotation Count	Data-abstraction Module
Current Speed	Current Speed Determine Speed Last Time	Data-abstraction Module DAM Update Operation Local Memory
Distance	Cumulative Distance Determine Distance Last Distance	Data-abstraction Module DAM Update Operation Local Memory
MPH	Initial Distance & Time Initialize MPH Compute Average MPH	Data-abstraction Module DAM Update Operation DAM Read Operation
MPG	Initial Distance & Fuel Level Initialize MPG Compute Average MPG	Data-abstraction Module DAM Update Operation DAM Read Operation

Table 39. Summary of CODA's Module Structuring Decisions (cont.)

Modules	Transformation/Data Store	Structuring Criterion
Oil Filter Maintenance	Miles at Last Oil Filter Maintenance Initialize Oil Filter Check Oil Filter Maintenance	Data-abstraction Module DAM Update Operation DAM Read Operation
Air Filter Maintenance	Miles at Last Air Filter Maintenance Initialize Air Filter Check Air Filter Maintenance	Data-abstraction Module DAM Update Operation DAM Read Operation
Major Service Maintenance	Miles at Last Major Service Initialize Major Service Check Major Service Maintenance	Data-abstraction Module DAM Update Operation DAM Read Operation
Cruise Control	Cruise Control	State-transition Module
Calibration Control	Calibration Control	State-transition Module
Maintenance Display	Maintenance Display	Device-interface Module
Mileage Display	Mileage Display	Device-interface Module
Throttle	Throttle	Device-interface Module
Clock	Clock	Device-interface Module
Gas Tank	Gas Tank	Device-interface Module
Shaft	Shaft	Device-interface Module
CC Lever	Cruise Control Lever	Device-interface Module
Calibration Buttons	Calibration Buttons	Device-interface Module
Maintenance Reset Buttons	Maintenance Reset Buttons	Device-interface Module
Mileage Reset Buttons	Mileage Reset Buttons	Device-interface Module
Brake	Brake	Device-interface Module
Engine	Engine	Device-interface Module
Calibration	Calibration Start Count Calibration Constant Record Calibration Start Compute Calibration Constant	Data-abstraction Module DAM Update Operation Multiple DAMs Combined By Designer

B.2.3 Integrating Tasks and Modules

Once task and module structuring are complete, the designer decides, for this case study, to ask CODA to integrate these two views. CODA first determines the logical placement of the twenty-five modules, relative to the eleven tasks. Device-interface modules for unshared devices are placed within the tasks that access the associated device; so, for example, the Brake module and the Engine module go inside the task named Monitor Auto Sensors and the Mileage Display module is placed inside the task named Compute Average Mileage. Modules accessed by a single task, such as Speed Control, which is accessed only by the task Control Auto Speed, are placed within the accessing task, while modules accessed by multiple tasks, such as Desired Speed, Current Speed, Clock, Gas Tank, Calibration, Distance, and Shaft Rotation Count, are placed outside any task.

After establishing module placement, CODA identifies cases where tasks invoke operations within modules that reside outside any task. In each such case, CODA establishes an Invokes relationship between the task and the operation. Additionally, CODA creates an Accesses relationship between a task and each module that provides operations invoked by that task. For example, the task named Determine Distance and Speed accesses two modules: 1) Distance, invoking the operation Update, and 2) Current Speed, invoking the operation Update.

Once the relationships between tasks and module are determined completely, CODA examines possible connections between modules residing outside any task. Where an operation in one such module invokes an operation in another such module,

CODA establishes a relationship stating that the invoking operation requires the invoked operation. For each module that provides operations required by another module, CODA creates a relationship indicating that the providing module serves the requiring module. For example, in this case study, an operation, Select, of the module Desired Speed, requires another operation, Read, provided by the module Current Speed. Current Speed, then, serves Desired Speed.

B.2.4 Defining Task Interfaces

All that remains to complete the design is the definition of task interfaces. At the designer's request, CODA begins this process.

B.2.4.1 Allocating External Task Interfaces

First, CODA determines the external interfaces for each task. At this point, CODA allocates data read and written by each task, determines timers and interrupts received by each task, and identifies the set of data and event flows exchanged among tasks. The details of these mappings are shown in Tables 40-44.

Table 40. CODA's Allocation of Input Data Flows to Tasks

Task	Input Data Flow
Compute Average Mileage	Fuel Amount Time of Day
Determine Speed and Distance	Time of Day
Monitor Auto Sensors	Brake Input Engine Input
Monitor Cruise Control Lever	Cruise Control Input
Monitor Reset Buttons	Fuel Amount Maintenance Reset Input Mileage Reset Input Time of Day
Perform Calibration	Calibration Input

Table 41. CODA's Allocation of Output Data Flows to Tasks

Task	Output Data Flow
Adjust Throttle	Throttle Position
Check Maintenance Need	Maintenance Display Data
Compute Average Mileage	Mileage Display Data

Table 42. CODA's Allocation of Interrupts to Tasks

Task	Interrupt
Monitor Cruise Control Lever	Lever Interrupt
Monitor Shaft Rotation	Shaft Interrupt

Table 43. CODA's Allocation of Timers to Tasks

Task	Timer
Adjust Throttle	Throttle Output Timer
Check Maintenance Need	Air Filter Timer Major Service Timer Oil Filter Timer
Compute Average Mileage	MPG Timer MPH Timer
Control Auto Speed	Increase Timer Resume Timer Speed Timer
Determine Speed and Distance	Distance Timer
Monitor Auto Sensor	Brake Sensor Timer Engine Sensor Timer
Monitor Reset Buttons	Button Poll Timer
Perform Calibration	Button Poll Timer

Table 44. Inter-Task Exchanges Identified by CODA

Destination Task	Source Task	Internal Data/Event Flow
Adjust Throttle	Control Auto Speed	Throttle Value (3 instances)
Control Auto Speed	Cruise Control	E/D Increase Speed E/D Maintain Speed E/D Resume Speed
Control Cruising	Control Auto Speed	Reached Cruising
	Monitor Cruise Control Lever	Accel Cruise Off Resume
	Monitor Auto Sensors	Brake Pressed/Released Engine On/Off

B.2.4.2 Allocating Control and Event Flows

Next, CODA considers how event flows between pairs of tasks might be allocated. CODA allocates event flows from the Monitor Auto Sensors and Monitor Cruise Control Lever tasks to queued messages. These events flow into a state-transition diagram and, thus, none should be missed and their arrival order should be preserved. In addition, the two input tasks that generate these events should not be delayed waiting for the Control Cruising task to accept the events.

CODA maps all control flows from the Control Cruising task to the Control Auto Speed task onto a single, tightly-coupled message. CODA makes this mapping because the Enable and Disable signals are assumed to be transmitted during a state-transition, and thus the sending task requires synchronization with the task receiving these control flows.

CODA is less certain how to map the event, Reached Cruising, that flows from the task Control Auto Speed to the task Control Cruising. In general, this decision depends upon whether the sender of the event needs to synchronize with the receiver of the event. CODA cannot determine if this is the case, and so, had the designer been inexperienced, then CODA would make a default decision to allocate this event to a queued message. For this case study, however, CODA consults the experienced designer. CODA asks the designer whether synchronization is required for this event. In this case, the designer says synchronization is not required, so CODA maps the event onto a queued message.

B.2.4.3 Allocating Data Flows

After deciding how to map all the events that flow between tasks, CODA next considers how to map all the data flows between pairs of tasks. In this case study, only three data flows, all instances of Throttle Value, to the Throttle must be considered. CODA, uncertain about the synchronization requirements for these data flows, consults the experienced designer for additional information. The designer indicates that the sender and receiver must rendezvous around these data flows; CODA then maps all three data flows to a single, tightly-coupled message from the task Speed Control to the task Adjust Throttle. Had an experienced designer been unavailable then, by default, CODA would map these three data flows to a single, queued message.

B.2.4.4 Eliciting Message Priorities and Defining Queue Interfaces

Next, CODA recognizes that one task, Cruise Control, receives queued messages from multiple source tasks. Since the designer is experienced, CODA offers the designer an opportunity to assign varying priorities to these messages. In this case study, the designer declines the offer. CODA then examines the facilities available in the intended target environment and defines appropriate mechanisms for holding queued messages. Since the target environment provides message queuing services and since tasks exchange queued messages at a single priority, CODA allocates a first-in, first-out message queue for each task that receives queued messages.

B.2.4.5 Completing Task-Interface Definition

After defining queue interfaces, CODA offers the designer a chance to review the new design elements created during task-interface definition. In this case study, the designer accepts the offer and commences the review. For each task, CODA displays only the incoming interfaces, except that CODA displays each datum output with the task that generates that output. This approach ensures that the designer reviews each element only once.

B.2.5 The Completed Design

At this point, the design is complete. Configuring the design and evaluating the performance of the design go beyond the scope of the prototype implemented for this dissertation. CODA can, however, generate specifications and design histories for each task and module in the design. In addition, CODA can check the design for completeness against the elements from the data/control flow diagram, and also for consistency with the design meta-model.

B.2.5.1 Creating the Software Architecture Diagram

When the designer requests that the design be written, CODA constructs a specification and design history for each task and module. The design histories are too long, and the task and module specifications too numerous, to include in this already long exposition. Instead, Figure 45 reproduces one task behavior specification, for the task Perform Calibration, and Figure 46 shows one module specification, for the module Calibration. The designer can use the task and module specifications to create a

<p>TASK: Perform_Calibration</p> <p>A) TASK INTERFACE: TASK INPUTS: Event Inputs: 1) Timer_Expiration (Timer event) from Run-Time_System every 0.5 secs. Data Inputs: 1) Calibration_Input from Calibration_Push_Buttons TASK OUTPUTS: MODULES ACCESSED: 1) Calibration Invokes Start Invokes Stop MODULES CONTAINED: 1) Calibration_Buttons 2) Calibration_Control</p> <p>B) TASK STRUCTURE: Criteria: Periodic Device I/O Task Control Task Control Cohesion Sequential Cohesion</p> <p>Transformations: 1.3.2 Calibration_Control 1.3.1 Calibration_Buttons 1.3.4 Compute_Calibration_Constant 1.3.3 Record_Calibration_Start</p> <p>C) TIMING CHARACTERISTICS: Activation: Periodic - by timer every 0.5 secs. Execution Time Ci:</p> <p>D) CONFIGURATION INFORMATION: Cardinality: 1 Priority: 1 Processor: 1</p> <p>E) TASK EVENT SEQUENCING:</p> <p>F) ERRORS DETECTED:</p>

Figure 45. Task Behavior Specification for Perform Calibration

MODULE: Calibration

A) MODULE LINKAGES:

Accessing Tasks: Perform_Calibration

Modules Served: Distance

B) MODULE STRUCTURE:

Criteria: Data-Abstraction Module

Update Operation Of A Data-Abstraction Module

Multiple, Data-Abstraction Modules Combined by Designer

Transformations: 1.3.3 Record_Calibration_Start

1.3.4 Compute_Calibration_Constant

Data Stores: Calibration_Start_Count

Calibration_Constant

C) ASSUMPTIONS:

This Module Supports Shared Access By Multiple Tasks

D) OPERATIONS PROVIDED:

1) Read_Constant

Output Parameter: Calibration_Constant

2) Start

Operation Required: Shaft_Rotation_Count.Read

3) Stop

Operation Required: Shaft_Rotation_Count.Read

Figure 46. Module Specification for Calibration

diagrammatic representation of the design. A two part figure, shown as Figures 47 and 48, gives such a representation of the concurrent design created for this case study. The task and module specifications exhibited in Figure 45 and Figure 46 are used below to show how a designer can map from the specifications to a diagram. The notation used in Figures 47 and 48 is explained in Chapter 5 of this dissertation.

Figure 47 illustrates diagrammatically the concurrent design, as generated by CODA, for the cruise control subsystem. The task behavior specification in Figure 45 applies to the Perform Calibration task, depicted in the upper portion of Figure 47, just to the right of center. The module specification in Figure 46 applies to the Calibration module, shown in Figure 47, just to the southeast of the Perform Calibration task. As revealed in the task behavior specification, Perform Calibration receives a timer event, Timer Expiration, and reads one input, Calibration Input. The task produces no direct outputs. The task does, however, access the Calibration module, invoking two operations, Start and Stop. The task also contains two modules, Calibration Control and Calibration Buttons. This information is represented pictorially in Figure 47. The task behavior specification includes additional information not shown on the diagram. For example, the timer period is 1/2 second and the task is a periodic device-i/o task, formed from four transformations based on three task-structuring criteria.

As revealed in the module specification, the Calibration module is accessed by one task, Perform Calibration, which uses the two operations, Start and Stop, and serves one module, Distance, which accesses the operation named Read Constant. This

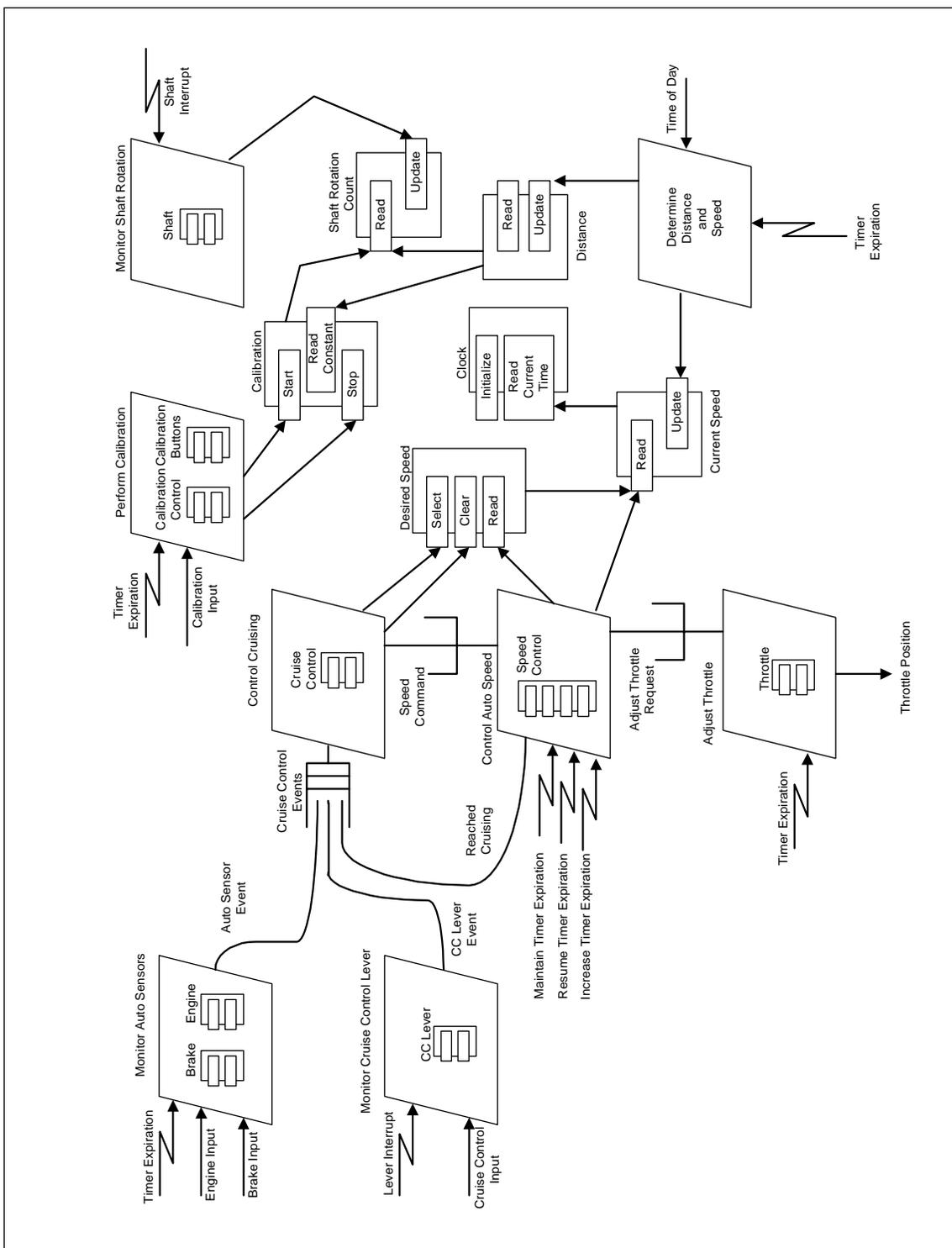


Figure 47. Automobile Cruise Control and Monitoring System Design (Part One of Two)

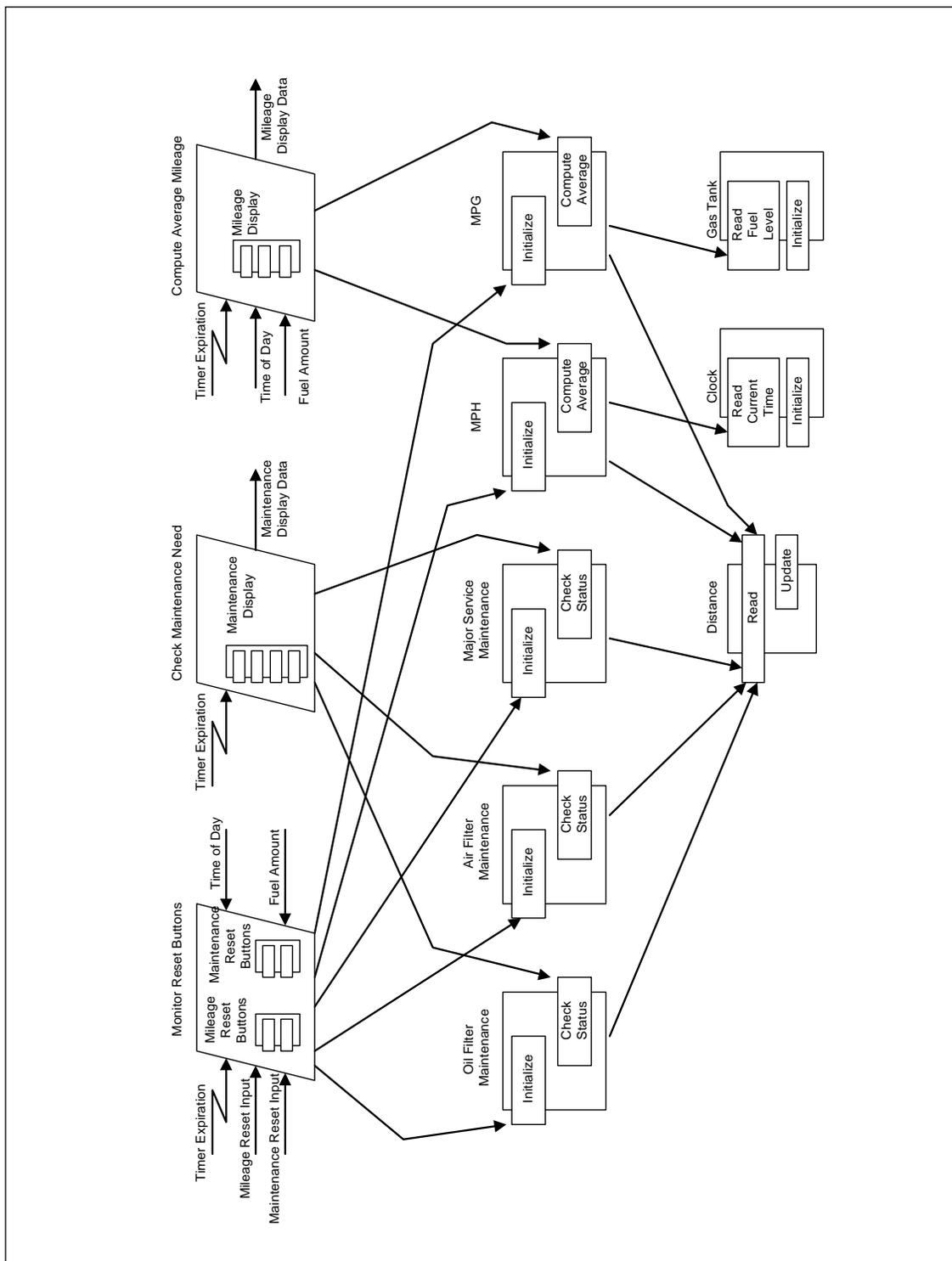


Figure 48. Automobile Cruise Control and Monitoring System Design
(Part Two of Two)

information, and the fact that the module is accessible from multiple threads of control, can be determined from reviewing the software architecture diagram, Figure 47. The module specification includes information not shown in Figure 47. For example, the module is formed from two data transformations and two data stores, based on three module-structuring criteria. In addition, one operation, Read Constant, returns a parameter, Calibration Constant.

The foregoing discussion should convince the reader that the contents of the software architecture diagram can be derived from the task behavior specifications and module specifications produced by CODA. In fact, the author derived Figures 47 and 48 using exactly that method.

Figure 48 depicts the software architecture for the monitoring subsystem. Two modules, Clock and Distance, appear on both Figures 47 and 48. Each of these modules is accessed by tasks from both subsystems. Replicating the modules on both figures provides a convenient means of viewing the design.

B.2.5.2 Assessing the Design

The concurrent design depicted in Figures 47 and 48 is almost identical to the design given by Gomaa for the automobile cruise control and monitoring system. [Gomaa93, Chapter 22] Two main differences can be discerned. First, one module, Clock, that appears in Figures 47 and 48 does not appear in Gomaa's design because Gomaa assumes a clock function is built into the operating environment. The time-of-day clock used in the data/control flow diagrams depicted earlier in this appendix is represented as an external

device; thus, CODA allocates a module to interface to that device. A second, more subtle, difference involves interactions between a task, Determine Distance and Speed, and two modules, Distance and Current Speed. In Gomaa's design, Determine Distance and Speed calls the Update operation in the Distance module and then the Update operation in the Current Speed module, just as shown in Figure 47. However, in Gomaa's solution, the Update operation in the Current Speed module invokes another operation, Read Incremental Distance, in the Distance module to obtain some information needed to compute the current speed. In the design generated by CODA, Determine Distance and Speed first invokes the Update operation in the Distance module. The update operation returns an output parameter, Incremental Distance, that is passed, by Determine Distance and Speed, as an input parameter to the Update operation in the Current Speed module. This difference between CODA's design and Gomaa's design results from the fact that CODA adopts a single strategy for mapping calls from tasks to modules, whereas, Gomaa uses a range of different strategies to perform these mappings.

B.3 Design Generated for a Novice Designer

To demonstrate design generation for a novice designer, CODA generates a second design for the automobile cruise control and monitoring system. The starting point for this design is the output from CODA's specification analyzer. This means that the input data/control flow diagram is identical to that used to generate the previous design, that is, the specification is fully classified, the axioms are satisfied, the timer values are identical, and the specification addenda are the same.

B.3.1 Generating the Design

In general, CODA moves through the same design-generation steps described previously in section B.2; however, in cases where an experienced designer was consulted, CODA now makes default decisions. The first such case occurs when CODA considers task mergers. Previously, two internal periodic tasks, one with a period of one second and another with a period of two seconds, were referred to the designer to consider combining them based on temporal and functional cohesion. For a novice designer, CODA simply refuses to consider such decisions; thus, by default, CODA does not combine the tasks in question (refer back to Table 35). Since, for the previous design, the experienced designer chose not to combine these tasks, CODA generates an identical result in each case.

Two other relevant cases appear during module structuring. First, CODA previously consulted the experienced designer regarding two isolated data stores, Last Distance and Last Time (refer back to section B.2.2.3). For a novice designer, CODA takes a default decision that these data stores provide local storage and then allocates these data stores to existing modules. Since, for the previous design, the experienced designer indicated that these data stores provide local storage, CODA generates an identical result in each case. A different outcome occurs with regard to module subsumption.

Previously, CODA consulted with the experienced designer regarding two cases where data-abstraction modules might be combined (refer back to section B.2.2.4). When

only a novice designer is available, CODA refuses to consider such complex cases; thus, by default, CODA does not combine modules. Since, for the previous design, an experienced designer chose to combine two modules, CODA generates a different result for the novice designer. This difference leads to other differences, as explained later, regarding module placement and module calling sequences.

Only two other instances arise where CODA wishes to consult an experienced designer. In one instance (refer back to section B.2.4.2), CODA desires to know the synchronization requirements for an event flow, Reached Cruising. Since the designer is a novice, CODA takes a default decision to map this event flow to a queued message. This decision agrees with the guidance provided by the designer in the previous design. In the other instance (refer back to section B.2.4.3), CODA wishes to know the synchronization requirements for three data flows, Throttle Value. Again, since the designer is a novice, CODA takes a default decision to map these data flows to a queued message. This decision differs from the guidance provided by the designer in the previous design.

B.3.2 The Completed Design

Figures 49 and 50 depict the design, generated by CODA for a novice designer. Figure 49 gives the design for the cruise control subsystem. Only minor differences exist between this solution and the solution generated with the assistance of an experienced designer (see Figure 47). One difference appears at the interface between the Control Auto Speed task and the Adjust Throttle task. In the previous design, the designer

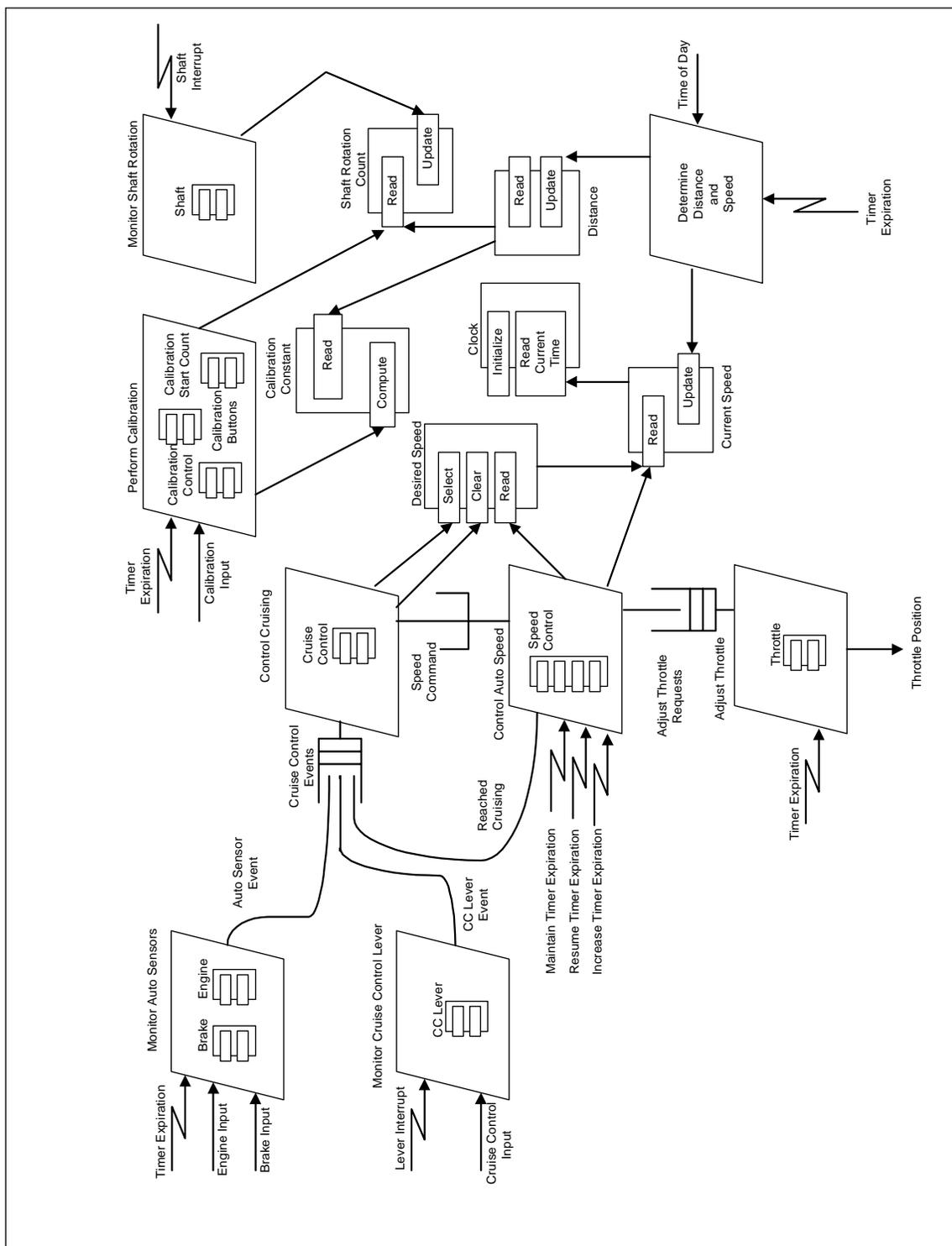


Figure 49. Cruise Control Subsystem Design - Novice Designer

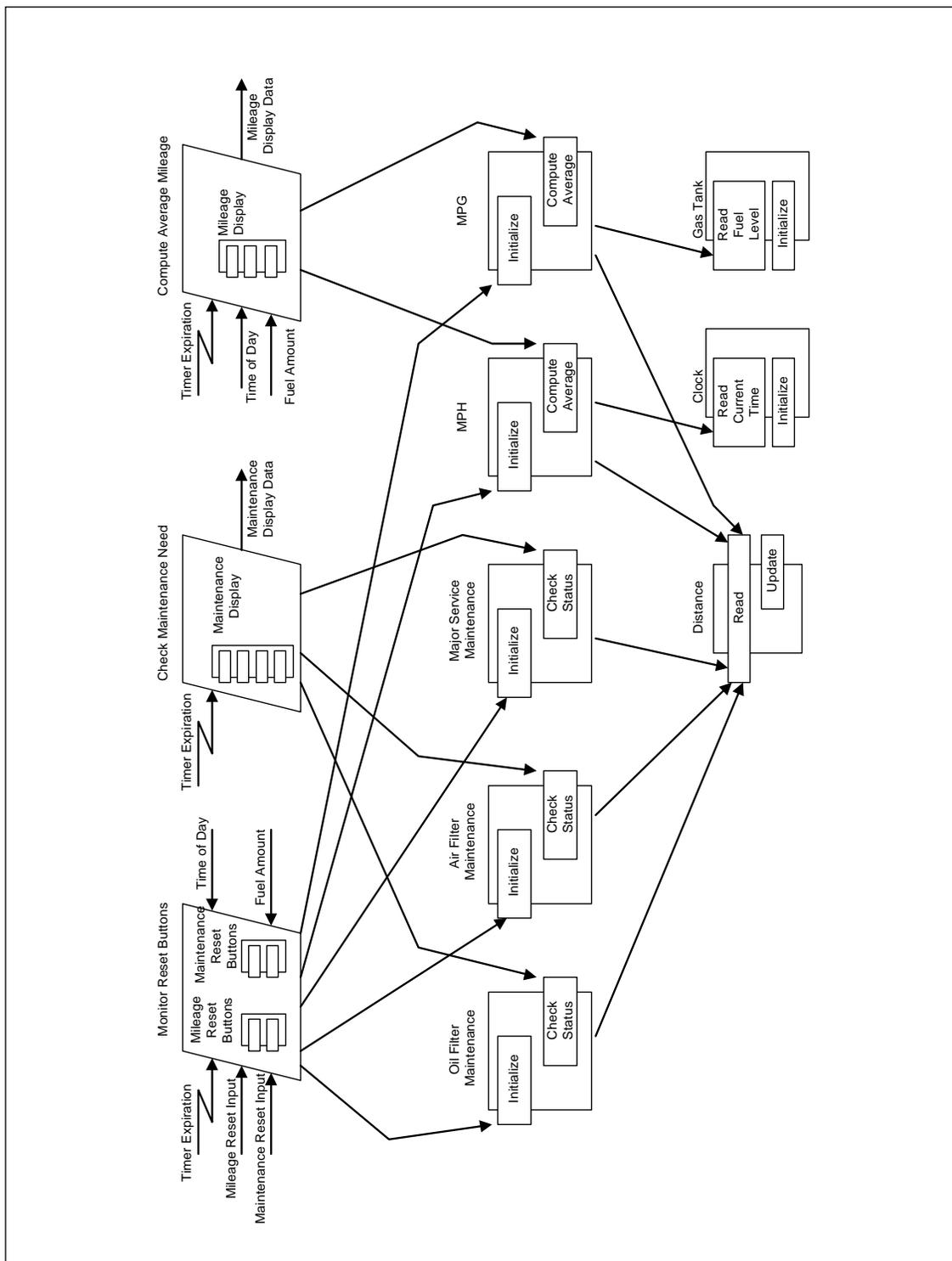


Figure 50. Monitoring Subsystem Design - Novice Designer

informed CODA that this message interface requires synchronization. Without such information, CODA generates a queued message interface by default. The other differences result from CODA's inability to determine that the Calibration Start Count and the Calibration Constant should be combined into a single data-abstraction module. For the previous design, the experienced designer advised CODA to combine the two modules. Without help from an experienced designer, CODA opts to leave the two modules separate. As a result, CODA places one of the modules, Calibration Start Count, inside of the Perform Calibration task and also shows an invocation from that task to the Read operation of the Shaft Rotation Count module. As a secondary effect, CODA assigns only two operations to the other module, Calibration Constant. For the remainder of the cruise-control subsystem design that CODA generates for the novice designer, the results appear identical to those generated by CODA with help from an experienced designer.

Figure 50 shows the design for the monitoring subsystem, as generated by CODA for a novice designer. No difference can be found between this design and the comparable design generated by CODA with help from an experienced designer (see Figure 48).