Object-Oriented Graphics for Requirements Analysis and Logical Design

Donald Firesmith, President
Advanced Software Technology Specialists
17124 Lutz Road
Ossian, Indiana, USA 46777
(219) 639-6305
(219) 747-9389 fax

Numerous graphic notations have been proposed during the last 5 years for use during Object-Oriented Requirements Analysis and Logical Design. The following article describes the consistent set of tool-supported object-oriented graphics that are part of ASTS Development Method 2 (ADM_2), a full development cycle method in use on various projects in the United States, Canada, and England.

1) Background

The original object-oriented graphic notation was language-specific and therefore developed for physical design [BO 1983]. Since then, several notations have been developed or adapted for use during object-oriented requirements analysis and logical design (OORALD). See, for example [BA 1989], [BO 1991], [CB 1989], [CM 1989], [CY 1989], [FG 1991], [RB 1991], [SM 1988]. This paper describes a useful object-oriented graphic notation that was developed as part of the ASTS Development Method 2 (ADM_2) to provide relatively complete, consistent, user-friendly diagrams that provide significantly more information than most other notations. This notation is currently being used on various projects in the United States, Canada, and England and has even been mandated by the Canadian Department of National Defense (DND) on the "Evaluation of Ada Compilers and Run-Time Executives (RTE) for the Militarized Reconfigurable Multiprocessor (MRM)" project. The following sections describe the basic notation, the five OORALD models to be supported by the associated diagrams, the five diagrams and their uses, and current tool support.

2) The Basic Notation

Various subsets of the following basic notation are used for General Semantic Nets (GSNs), Classification Diagrams (CLDs), Composition Diagrams (CMDs), Object-Interaction Diagrams (OIDs), part of object-oriented State Transition Diagrams (STDs), and object-oriented Control Flow Diagrams (CFDs). See Figure 1: Primary Node Icons.

The primary entities of object-oriented requirements analysis and logical design (and thus the primary icons of the ADM_2 graphic notation) are objects, classes, and metaclasses. Because they are encapsulations protected by information hiding and because of their importance as entities, they are represented with thick borders to signify the protection of their encapsulated resources and for easy recognition on the diagrams that contain them. Objects are represented by a single icon, classes are represented by a double icon implying a multiplicity of objects in the class, and metaclasses are represented by a triple icon implying a multiplicity of classes in the metaclass. Separate icons have been developed for concurrent and sequential objects, classes, and metaclasses because concurrency issues are critical in many applications [SH 1991] and therefore must be considered from the beginning in order to correctly specify and design the correct client/server relationships and the exportation of operations by objects, classes, and metaclasses. Concurrent objects, classes, and metaclasses are represented by parallelograms implying parallel execution (at least in theory), while sequential objects, classes, and metaclasses are represented by rectangles implying software black boxes.

The concept of subassembly (a.k.a., subsystem or computer software component) is important because almost all non-trivial applications contain more than the Miller (i.e., Hrair) limit of seven plus or minus two [ML 1956] objects,

classes, and/or metaclasses. Subassemblies come in the following two important forms: recursive (if resulting from the recursive use of a recursive development method) and ad hoc (if otherwise). Subassemblies are represented as thick rounded rectangles (if recursive) or ovals (if ad hoc). The visibility of exported or hidden objects, classes, and metaclasses is signified by socketing the exported entities and nesting the hidden ones. Assemblies, as collections of subassemblies, are represented by thick double rounded rectangles implying a multiplicity of subassemblies.

Hardware is represented as a three-dimensional box implying a hardware black box, while a system is represented as a thick three-dimensional box implying more than just hardware.

While global operations and common global data are certainly not object-oriented and are in many ways antithetical to what object-orientation stands for, analysis and design notation must still deal with the possible need to document such entities. Although ADM_2 recommends avoiding their use, they are still covered in the basic notation as follows: concurrent and sequential global operations are represented by a single parallelogram and rectangle respectively, while common global data is represented by the traditional data store icon.

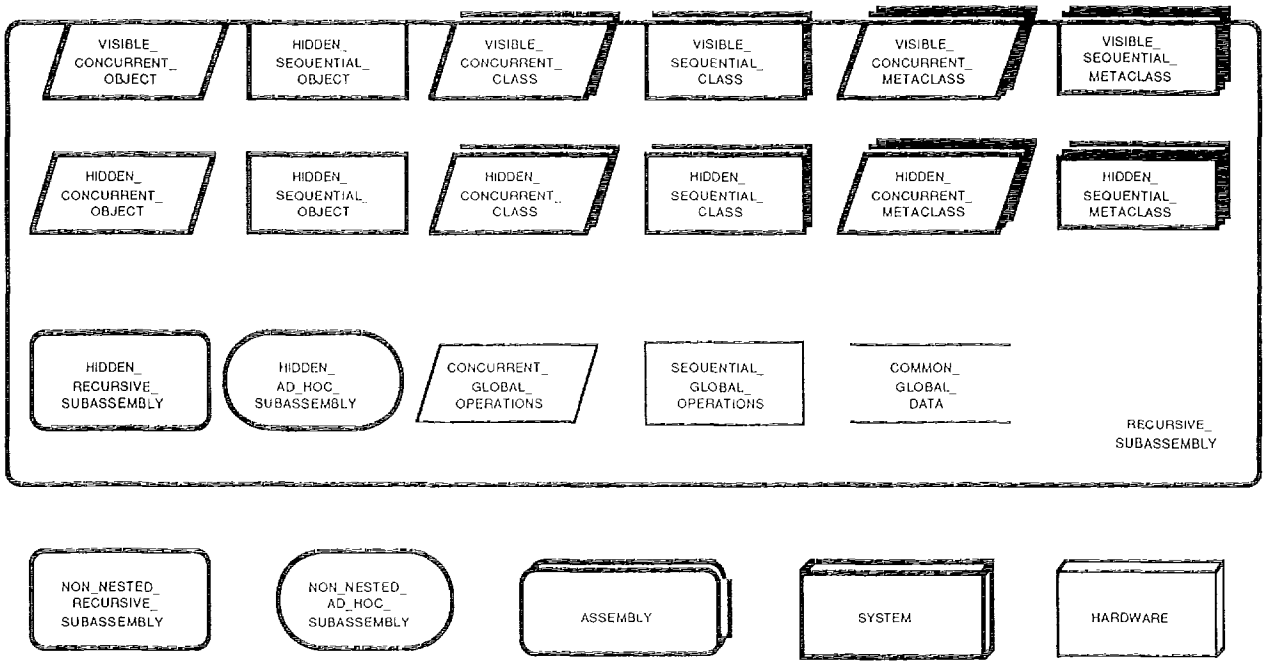

Figure 1: Primary Node Icons

3) Five Models of Object-Oriented Requirements Analysis and Logical Design

The five models of object-oriented requirements analysis and logical design addressed by ADM_2 are: the semantic model, the object-interaction model, the state model, the control model, and the timing model. These models are typically developed in this order and provide increasingly detailed views of the subassembly and its component objects, classes, and metaclasses being analyzed and designed.

The *Semantic Model* for each subassembly models: all essential and logical objects, classes, and metaclasses in a subassembly; the system, hardware, and software terminators with which the subassembly must interface; any non-object oriented entities such as global operations and data, and all important semantic relationships between them. The semantic model thus identifies the basic [object-oriented] entities and how they relate to one another. It therefore is a good starting model and useful for understanding the application and communication with non-software individuals involved with the project. The semantic model is documented with General Semantic Nets (GSNs) and the two specialized semantic nets: Classification Diagrams (CLDs) and Composition Diagrams (CMDs).

The *Object-Interaction Model* for each subassembly models: all essential and logical objects, classes, and metaclasses in a subassembly; the system, hardware, and software terminators with which the subassembly must interface; any global operations, and all exported operations that they require from one another. Because objects, classes, and metaclasses interact by calling one another's exported operations (i.e., sending messages to one another requesting the invocation of the associated method), the object-interaction model thus shows how essential and logical objects interact. The object-interaction model is documented with Object-Interaction Diagrams (OIDs).

The *State Model* for each essential and logical object, class, or metaclass models all states of the entity and the transitions between these states. The state model is documented with object-oriented State Transition Diagrams (STDs).

The *Control Flow Model* may exist at the subassembly, thread, or object/class/metaclass level. It models: all relevant essential and logical objects, classes, and metaclasses; the system, hardware, and software terminators with which they must interface; any relevant global operations; all exported and hidden operations (if useful) datastores; and the control flows (and data visibility relationships) between them. The control flow model is documented with object-oriented Control Flow Diagrams (CFDs).

The *Timing Model* may exist at the subassembly, thread, or object/class/metaclass level. It models the expected sequencing and/or time relationships of operation calls between objects, classes, metaclasses, system terminators, hardware terminators, software terminators, any relevant global operations, and all exported and hidden operations. The control flow model may be partially documented with object-oriented Timing Diagrams (TDs).

4) The Five Diagram Types

4.1) Semantic Nets (SNs)

General Semantic Nets (GSNs) use the primary icons as shown in Figure 1. The primary arc connecting these nodes is the named relationship that can represent any relationship important to the analysis or design. Three specific relationships that occur frequently are the "has subclass" and "has instance" relationships that make up inheritance hierarchies and the "has component" relationship that makes up composition hierarchies. These relationships are shown below in Figure 2: GSN Arc Icons.

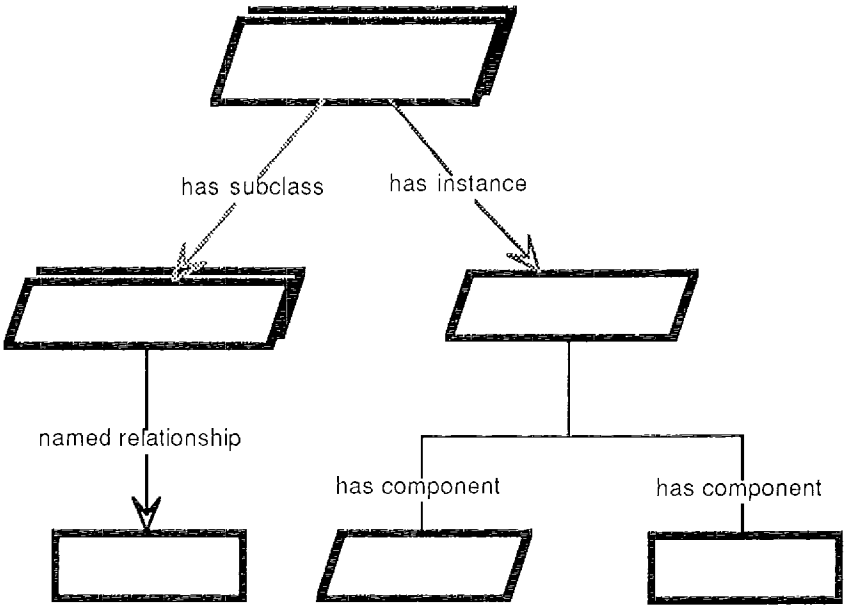

Figure 2: GSN Arc Icons

General semantic nets can be used as context diagrams. The following example context diagram uses the "has component" relationship to show a composition hierarchy in the hardware design. See Figure 3: Context Diagram for an Automotive Dashboard Application.

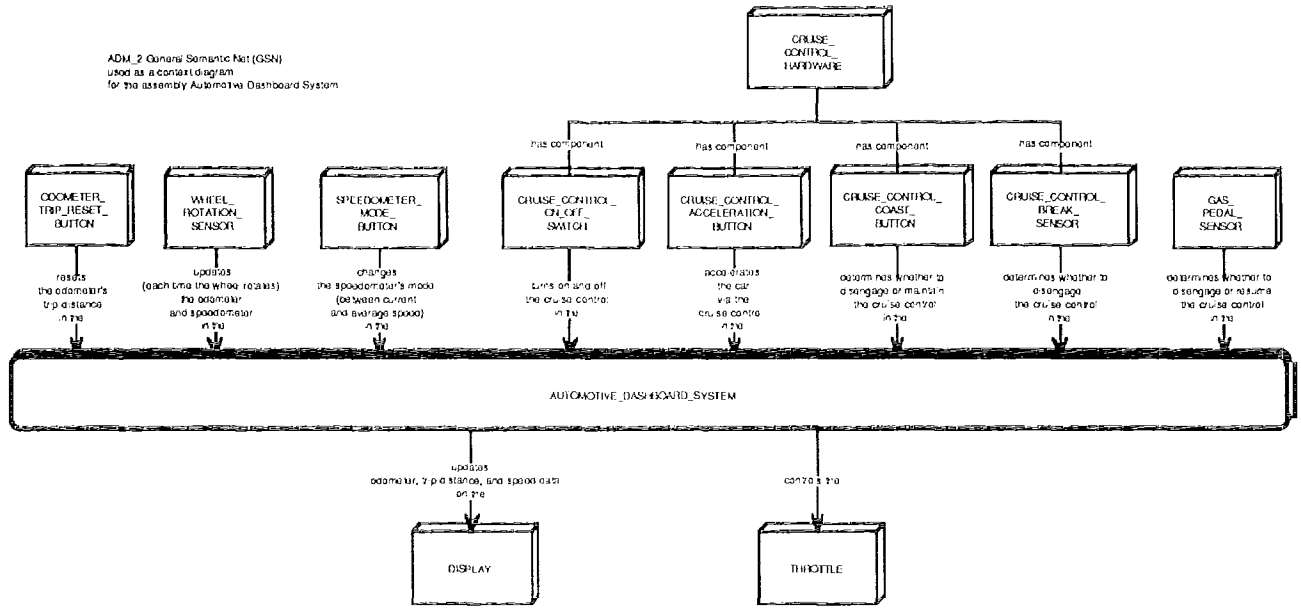

Figure 3: Context Diagram for an Automotive Dashboard Application

Each SN identifies the important semantic relationships between candidate essential and logical abstract objects, classes, and possibly other important entities. ADM_2 SNs provide more information than other semantic nets (e.g., the information model of [SM 1988]) and are more powerful and user friendly than Entity Relationship Attribute (ERA) or Chen Diagrams [PC 1977].

Consider the important ramifications (e.g., polling, interrupts) of concurrency decisions upon the relationships between objects, classes, and external entities with which they interface.

The CRUSE_CONTROL subassembly depicted in Figure 4 contains five exported concurrent master objects, one hidden concurrent agent object, and one hidden sequential slave object. Note also the dependency on the exported concurrent object SPEEDOMETER in the child subassembly DISPLAY.

Classification Diagrams (CLDs) are used to document inheritance hierarchies. Overview Classification Diagrams show the classification hierarchy in terms of superclasses, subclasses, and optionally instances. See Figure 5: Example Overview Classification Diagram. Detailed CLDs show superclasses, subclasses, and their specifications and bodies including what capabilities are added, modified, or deleted from subclasses. See Figure 6: Example Detailed Classification Diagram. Evaluate inheritance relationships carefully. Use inheritance to simplify related classes, but *not* merely to simplify the reuse of random unrelated capabilities that do not logically belong to objects of the same superclass. Avoid unnecessary inheritance coupling.

Composition Diagrams (CMDs) show the "has part" relationships and are useful to show the structure of aggregate classes and objects, (i.e., individual composition hierarchies). See Figure 7: Example Composition Diagram (CMD) for a Traffic Signal. When discussing components of classes, one must differentiate between components of the class as a whole and instances of the class.

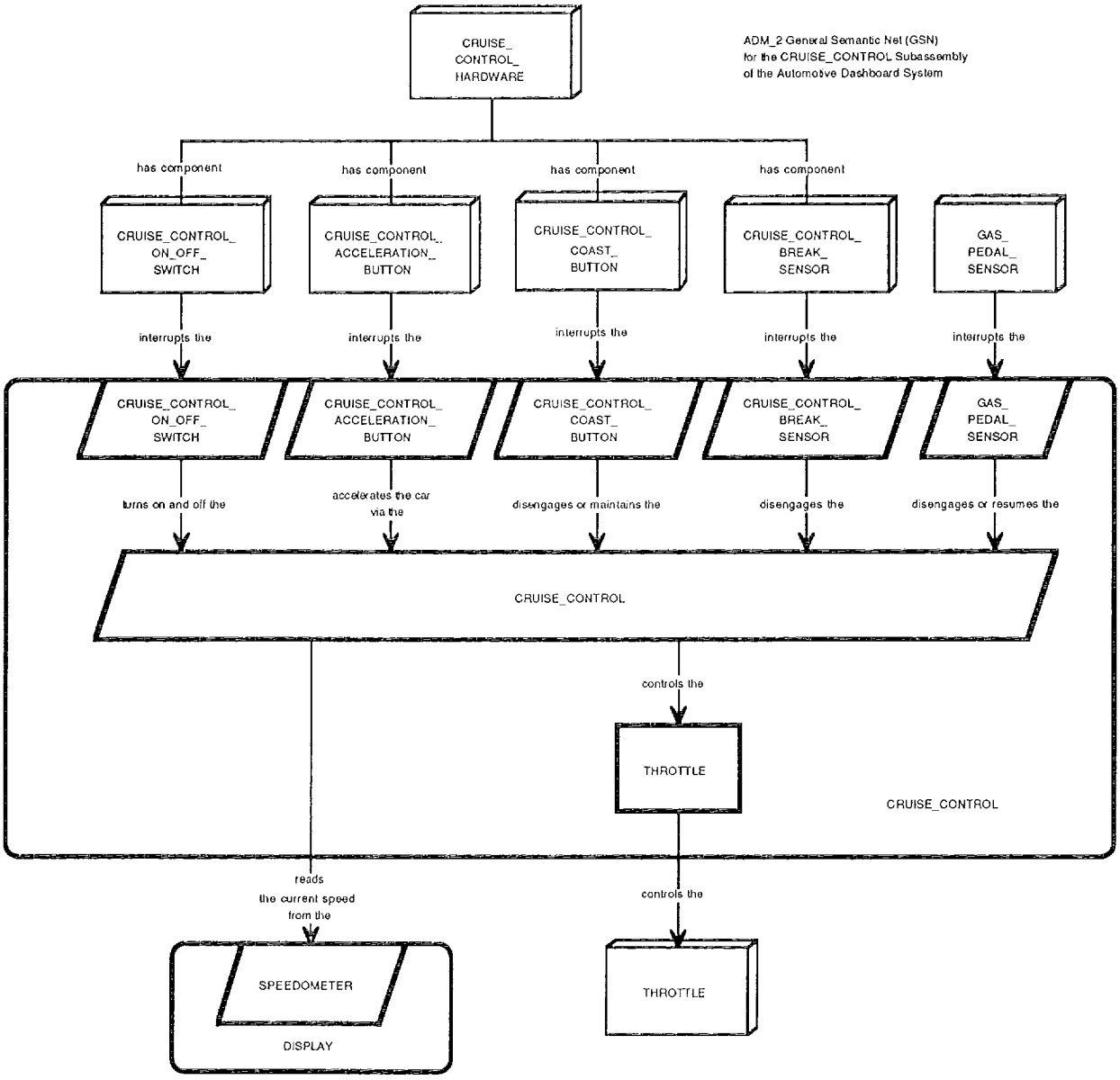

Figure 4: GSN for an Automatic Dashboard Application

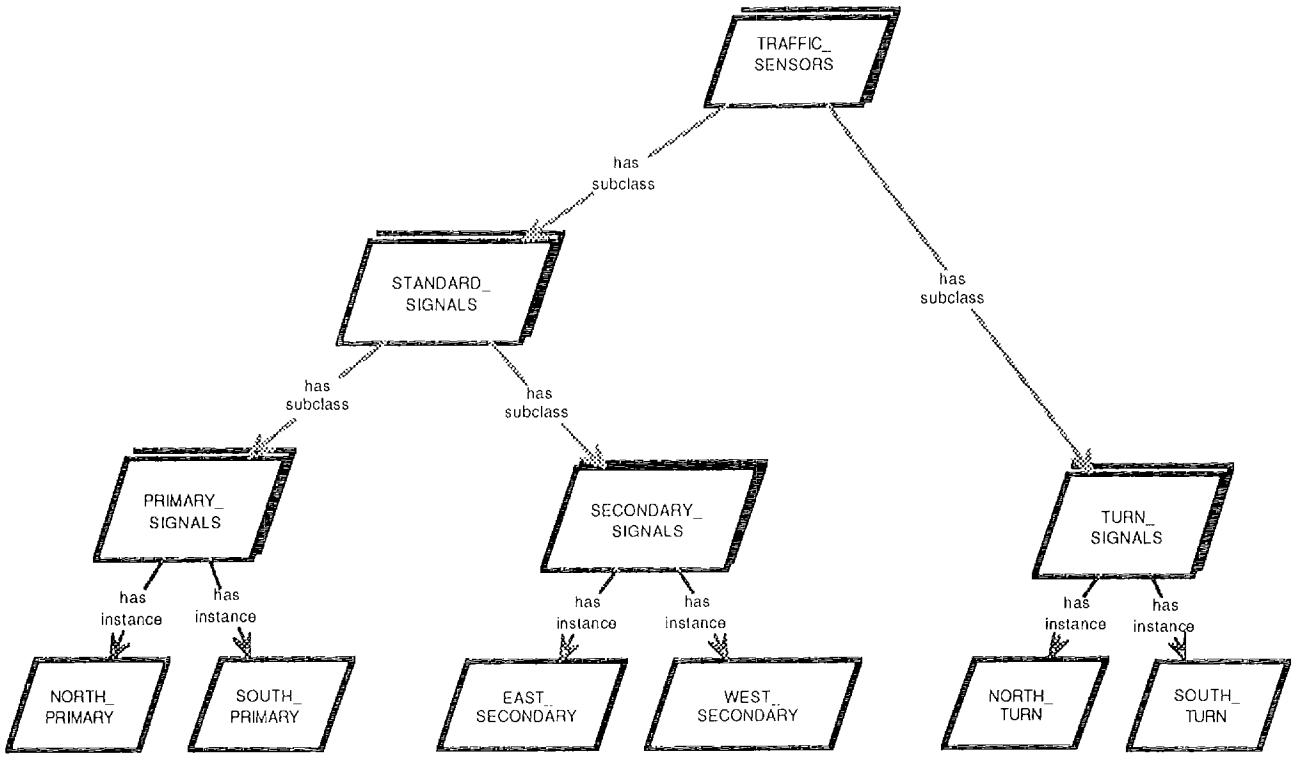

Figure 5: Overview Classification Diagram (CLD) for Traffic Sensors

4.2) Object-Interaction Diagrams (OIDs)

The Subassembly OID (SOID) is the primary overview diagram of subassembly requirements analysis and logical design. See Figure 8: OID for a Digital Thermostat System.

The OIDs expand upon the information in the Subassembly Semantic Nets, and, with proper tool support, the SOID can be generated automatically from the information in the Structured Descriptions.

Note that the required operation arcs do not necessarily list all exported operations of the object, class, metaclass, or external slave entity that they point to, but rather only those exported operations that are used to implement the relationship on the corresponding arc on the corresponding General Semantic Net. Note also that if two required operation arcs point to the same entity, they do not necessarily have to list the same set of exported operations.

ADM_2 OIDs were derived from Ed Colbert's OIDs [EC 1989].

4.3) Object-Oriented State Transition Diagrams (STDs)

Where cost-effective and useful, develop the *Subassembly State Model* using essential class and abstract object *State Transition Diagrams* (STDs). See Figure 9: State Transition Diagram (STD) Icons.

The object- or class-level STD documents the states of a single essential object class or abstract object, and the constructor and modifier operations and exceptions that cause or may conditionally cause state transitions.

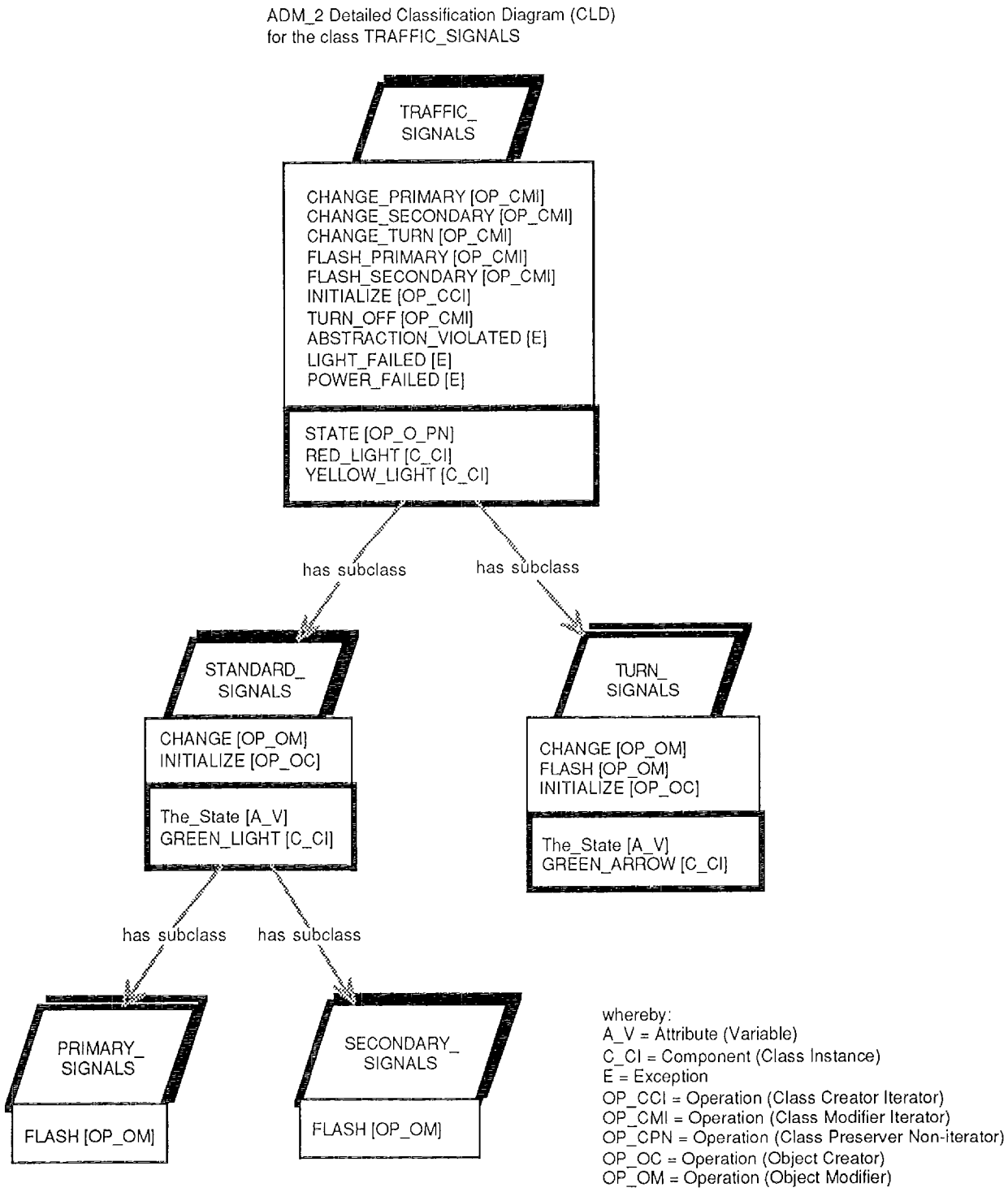

Figure 6: Example Detailed Classification Diagram

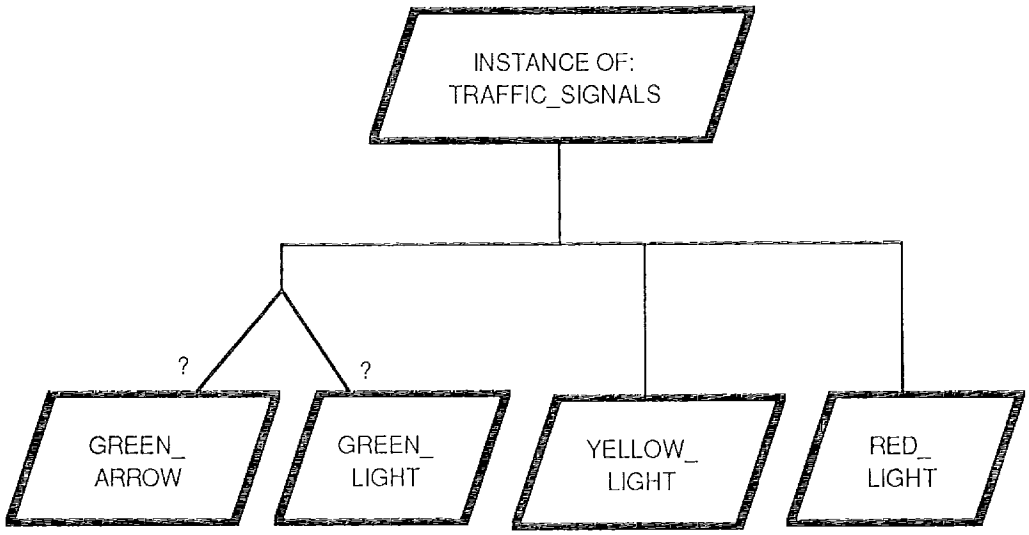

Figure 7: Example Composition Diagram (CMD) for a Traffic Signal

Constructor and modifier operations documented on STDs come in the following three varieties:

1) exported by the current class or object.
2) hidden within the current class or object.
3) exported by some other class, object, or external hardware or software.

If a constructor or modifier operation of the current class or object also causes another class or object to change its state, this should also be documented on the current STD.

When an event (such as the passing of time) triggers a transition, consider it a modifier operation on some other object such as CLOCK.DELAY_UNTIL or CLOCK.DELAY_FOR.

Note on Figure 10 that:

- the OFF state is the initial state.
- the ON state is a compound state that decomposes into the three substates ACCELERATING, DISENGAGED, AND MAINTAINING.

4.4) Object-Oriented Control Flow Diagrams (CFDs)

Develop detailed scenarios of both expected and error-handling threads of control between and within the objects and classes of the subassembly, and use these scenarios to analyze the associated operation calls and/or exception flows between and within the objects and classes. Use the scenarios to develop one or more *Object-Oriented Control Flow Diagrams* (CFDs).

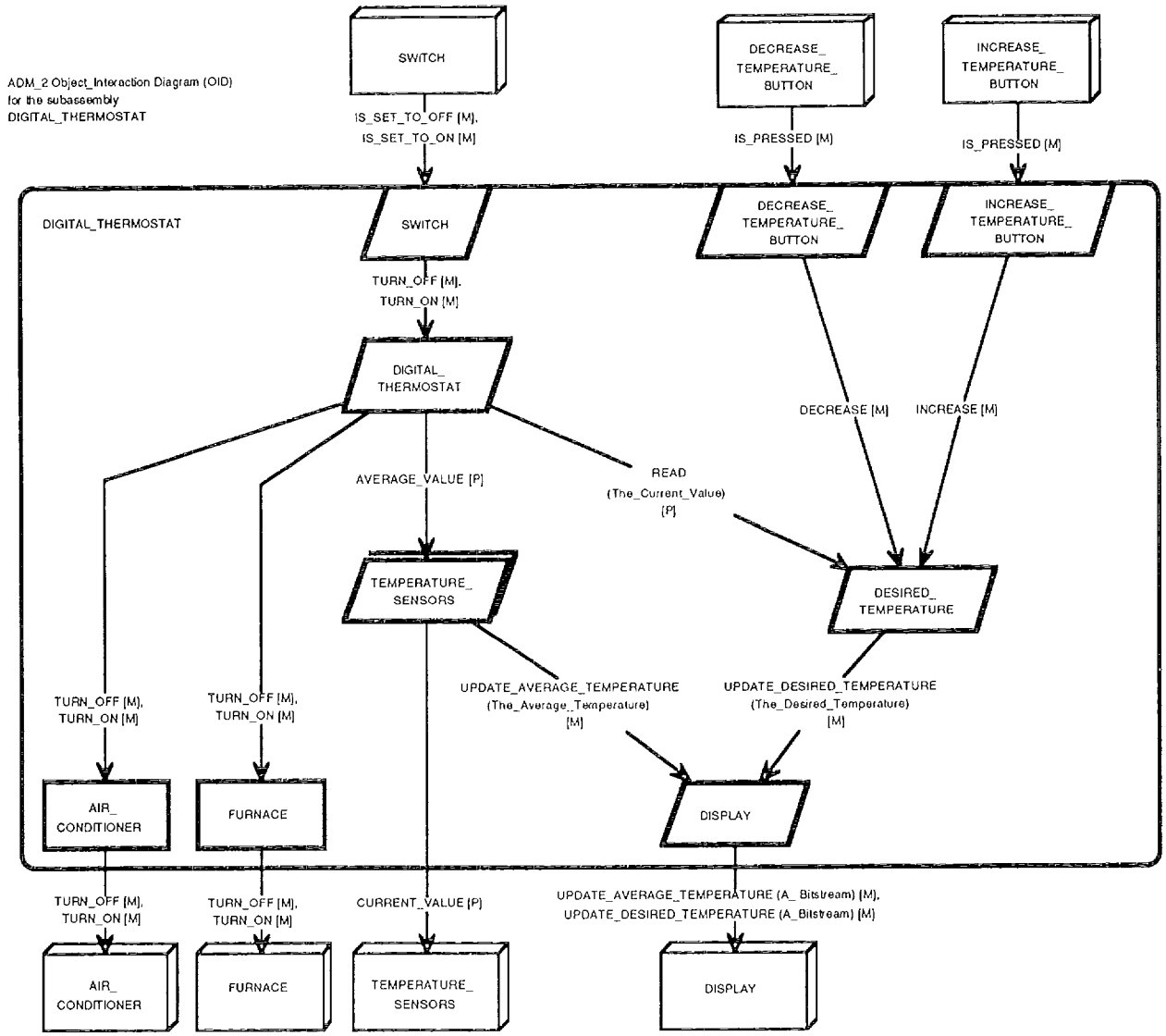

Figure 8: OID for a Digital Thermostat System

The scope of each CFD may be:

1) a single essential or logical object or class. See Figure 11: Example Object-Level Control Flow Diagram (CFD).

2) a thread of control across multiple objects and classes within the subassembly. See Figure 12: Example Thread-Level Control Flow Diagram (CFD), which shows the primary and secondary threads of control that cause the signals of a street intersection to flash.

3) an entire subassembly. See Figures 13: Example Subassembly Control Flow Diagram (CFD).

CFDs are to ADM_2's requirements analysis and logical design step what structure charts are to Structured Design.

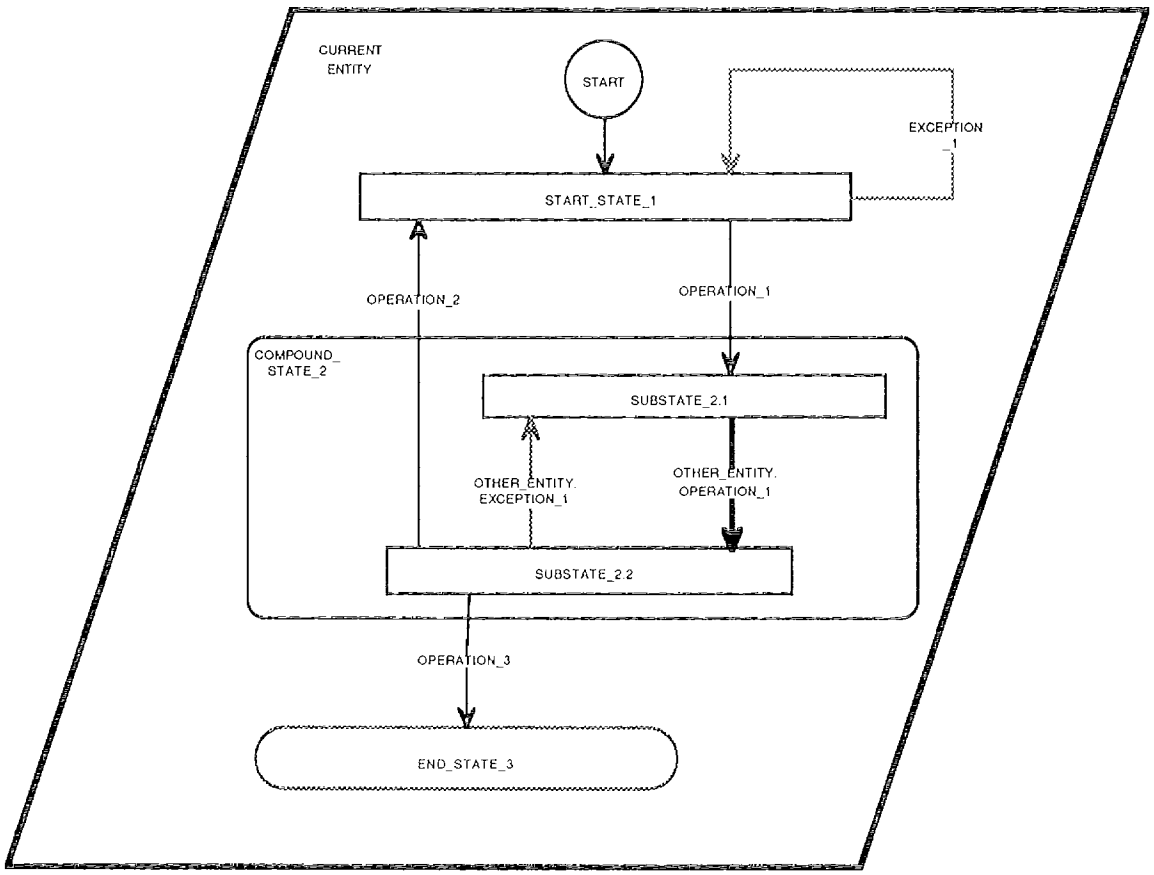

Figure 9: State Transition Diagram (STD) Icons

Where cost-effective, add data stores and data visibility arrows to the CFDs to make them also Object-Oriented Data Flow Diagrams (DFDs). Note that an object or class may have more than one data store if it has more than one attribute. Using data stores to identify objects and classes may therefore be misleading. The OO CFD/DFD is used to identify hidden operations (via transforms) and attributes (via data visibility arrows and stores).

4.5) Timing Diagrams

The Subassembly Timing Diagram (TD) is a modified Booch Timing Diagram [BO 1991], is object-oriented, and documents the subassembly behavior in terms of the timing of operation calls between the main subprogram, stubbed operation, or stubbed object and the remaining objects, classes, and metaclasses. Note that when useful, the state transitions of objects, classes, and metaclasses can also be shown, providing the sequential timing of transitions on State Transition Diagrams. See Figure 14. TDs may be used to show required, designed, or actual calling behavior. They may, however, be replaced by annotating the CFDs with timing information.

5) Tool Support

All of the above graphics are currently supported by the CASE tool ObjectMaker® version 1.8 developed by Mark V Systems Ltd. of Encino, California using menu and rule files developed and maintained by ASTS. Mark V Systems may be contacted at (800) 666-6Ada. ObjectMaker currently supports: 1) user-friendly drawing of the diagrams using pull-down menus of the valid icons, 2) interactive node parent rule checking to ensure than various nodes may only be socketed and/or nested in valid parent nodes, and 3) interactive arc rule checking to ensure that the various arcs may only be drawn from and to valid nodes. Work is currently in progress to complete the mapping to the semantic database. Future work is intended to include improving the user-friendliness of the Timing Diagram, cross-diagram checking, the automatic generation of skeleton code from OIDs and CFDs, and the generation of object-oriented requirements specifications and design documents from the database. The ADM_2 graphic notation is not proprietary, and ASTS is thus seeking to cooperate with other CASE tool vendors.

Acknowledgment

All except the last diagram used in this paper were generated using version 1.8 of the Objectmaker CASE tool developed by Mark V Systems Limited.

References

[BA 1989] Bailin, Sidney C. "An Object-Oriented Requirements Specification Method," *Communications of the ACM*, Volume 32, Number 5, May 1989, pages 608-632.

[BO 1983] Booch, Grady, *Software Engineering with Ada*, Benjamin/Cummings Publishing Company, 1983.

[BO 1991] Booch, Grady, *Object-Oriented Design with Applications*, Benjamin/Cummings Publishing Company, 1991.

[PC 1977] Chen, Peter, *The Entity-Relationship Approach to Logical Data Base Design*, Q.E.D. Information Scienc 1977.

[CB 1989] Colbert, Edward, "The Object-Oriented Software Development Method: A Practical Approach to Object Oriented Development," *Proceedings of TRI-Ada'89 -- Ada Technology In Context: Application, Development, and Deployment, 23-26 October 1989,* pages 400-415.

[CM 1989] Comer, Ed, *Ada Box Structures Methodology Handbook,* Software Productivity Solutions, (407) 984-3370, 31 July 1989.

[CY 1989] Coad, Peter and Yourdon, Ed, *OORA - Object-Oriented Requirements Analysis,* Prentice Hall, 1989.

[FG 1991] Firesmith, Donald and Gaumer, Dale, *The ASTS Development Method 2 (ADM_2) User's Manual,* ASTS, Fort Wayne, Indiana, (219) 639-6305, 1991.

[ML 1956] Miller, G. A., "The Magical Number Seven, Plus or Minus Two," *The Psychological Review*, Volume 63, Number 2, March 1956.

[RB 1991] Rumbaugh, James; Blaha, Michael; Premerlani, William; Eddy, Frederick and Lorensen, William. *Object-Oriented Modeling and Design,* Prentice Hall, 1991.

[SH 1991] Shumate, Ken, "Structured Analysis and Object-Oriented Design are Compatible," *Ada Letters*, Volume XI, Number 4, May/June 1991, pages 78-90.

[SM 1988] Shlaer, Sally and Mellor, Stephen J., *Object-Oriented Systems Analysis, Modeling the World in Data,* Yourdon Press, 1988.

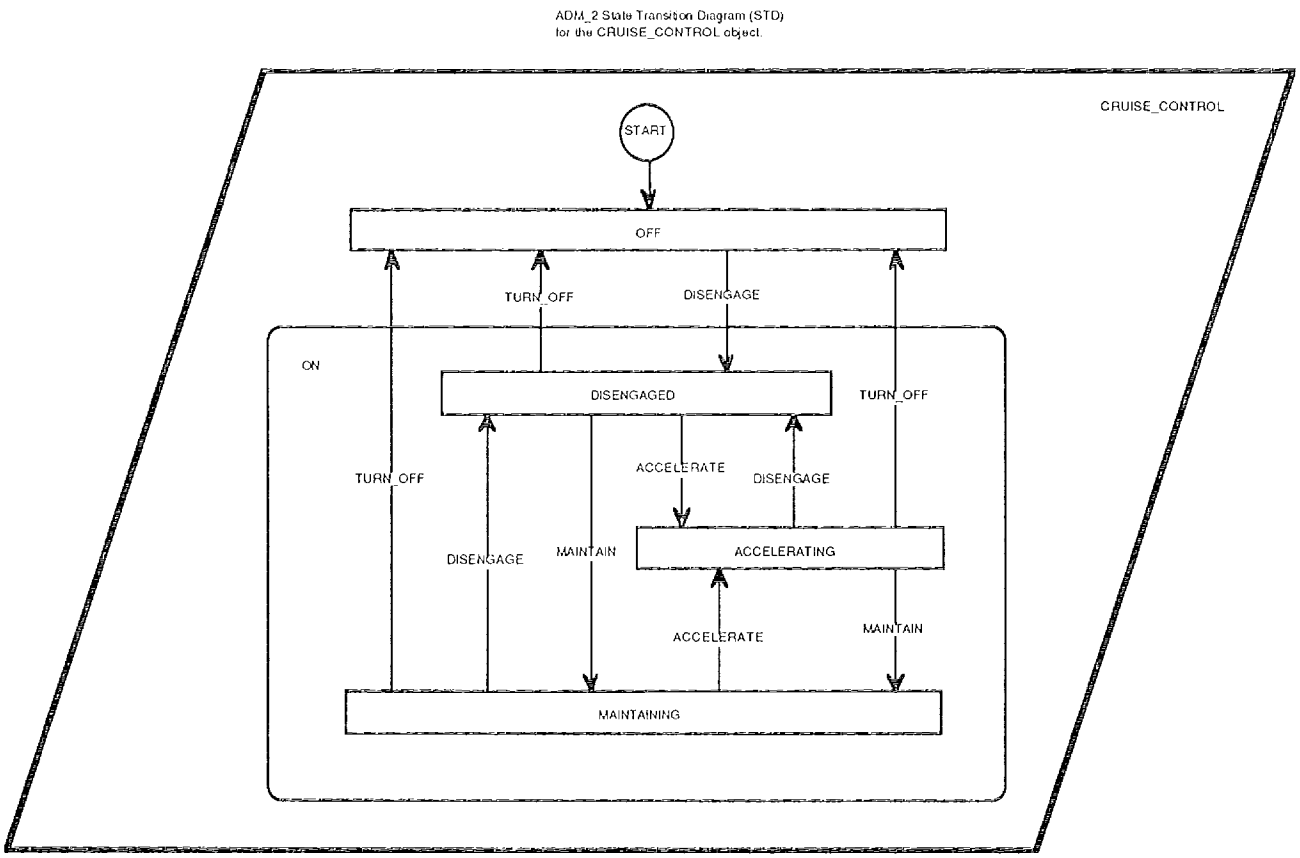

Figure 10: Example Object-Level State Transition Diagram (STD)

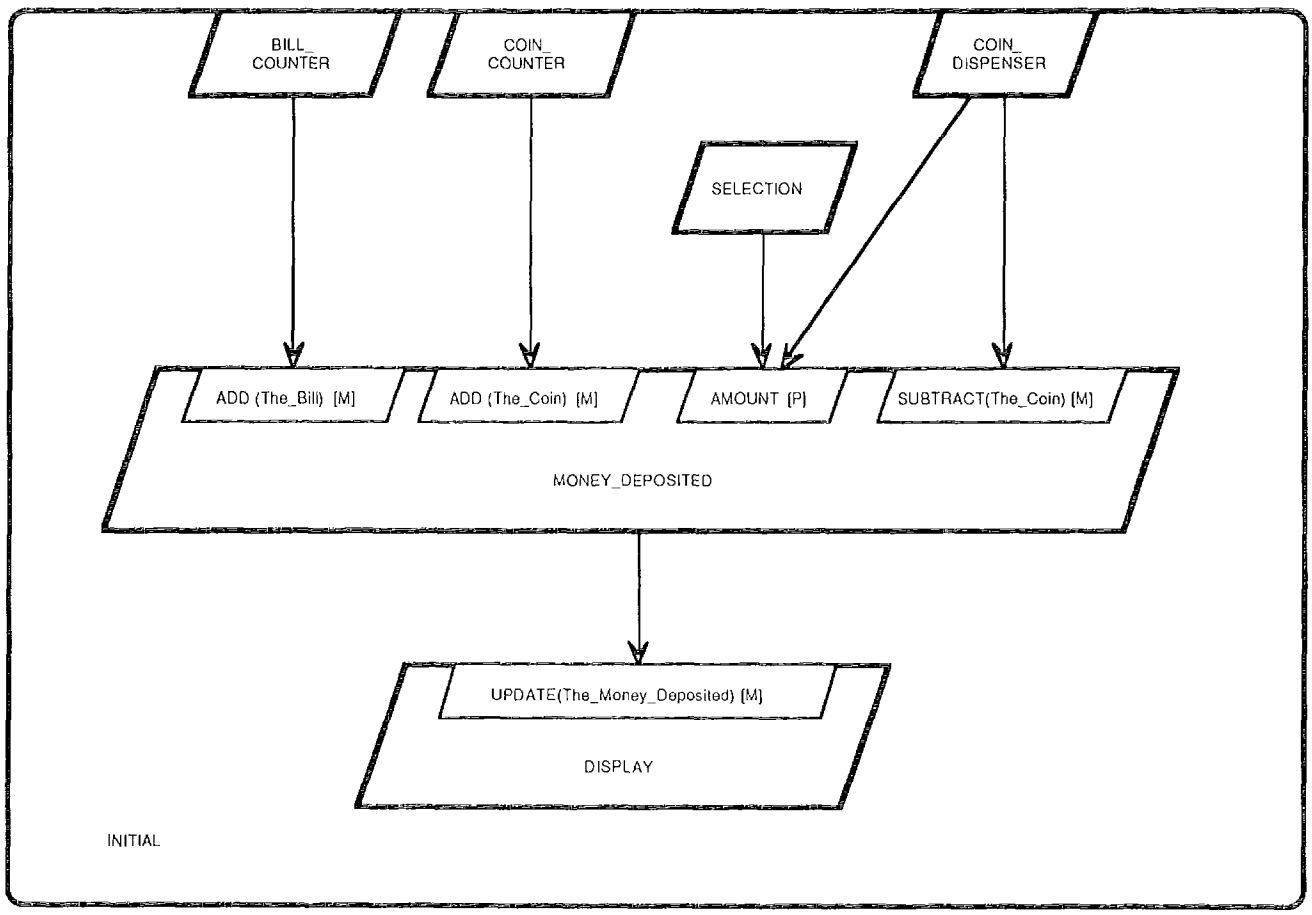

Figure 11: Example Object-Level Control Flow Diagram (CFD)

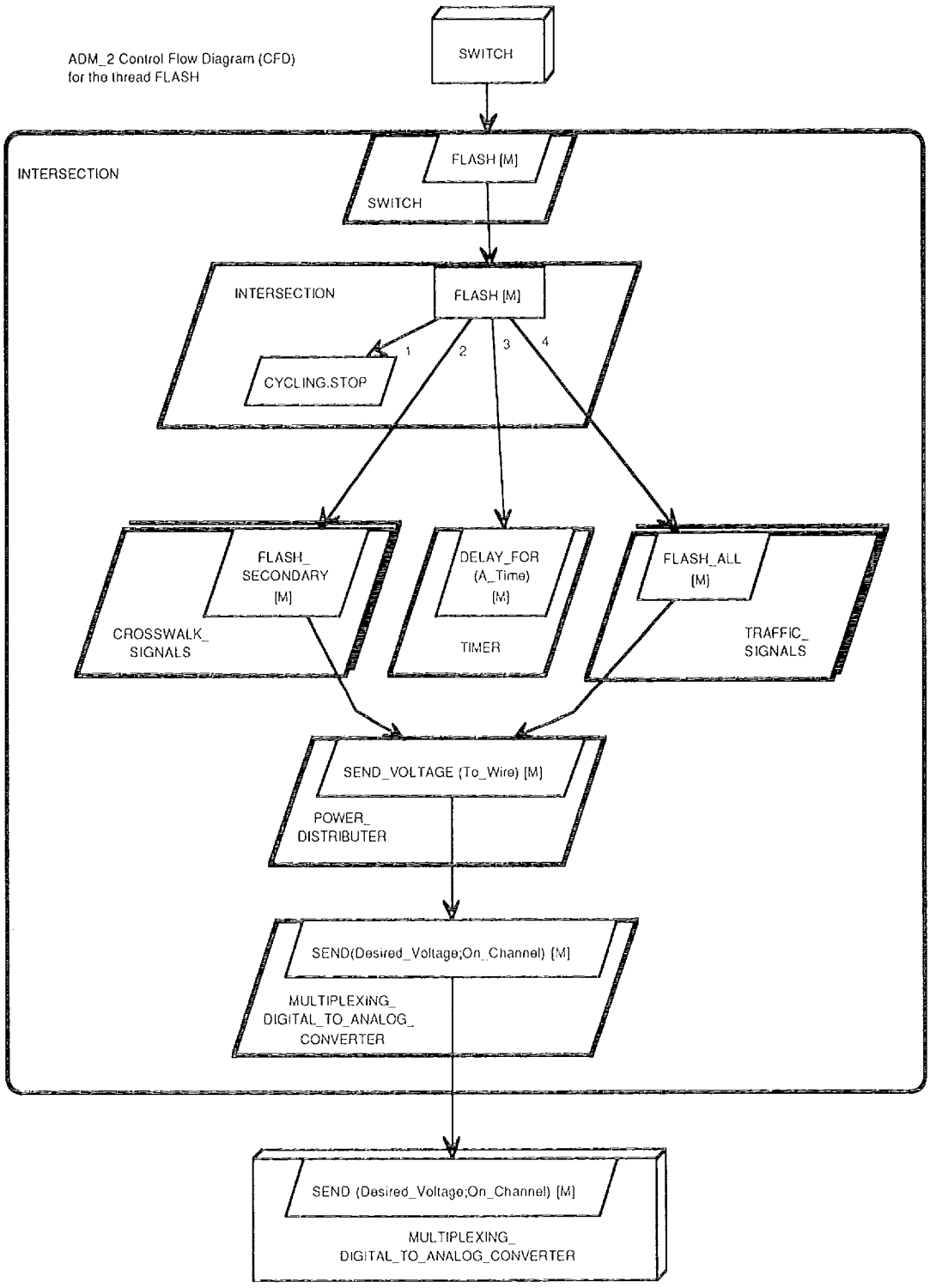

Figure 12: Example Thread-Level Control Flow Diagram (CFD)

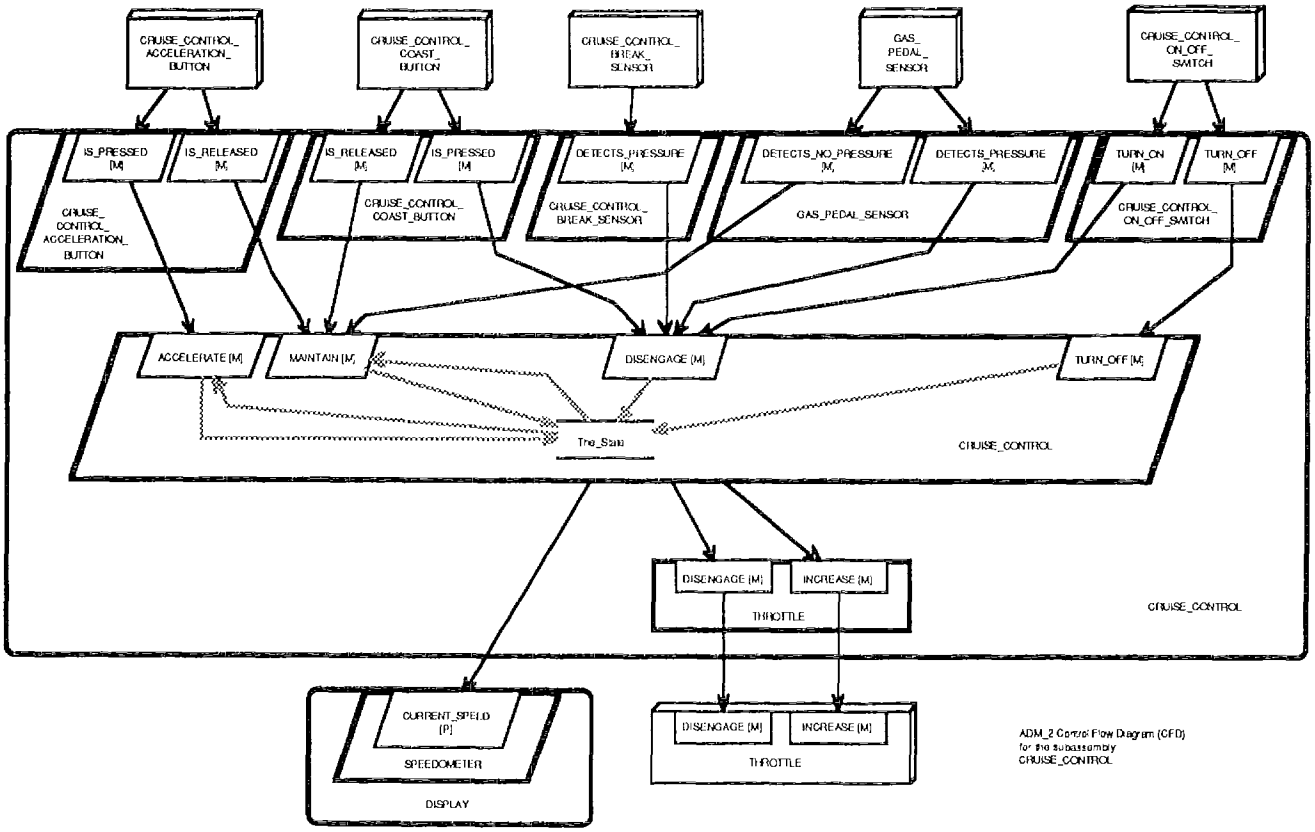

Figure 13: Example Subassembly-Level Control Flow Diagram (CFD)

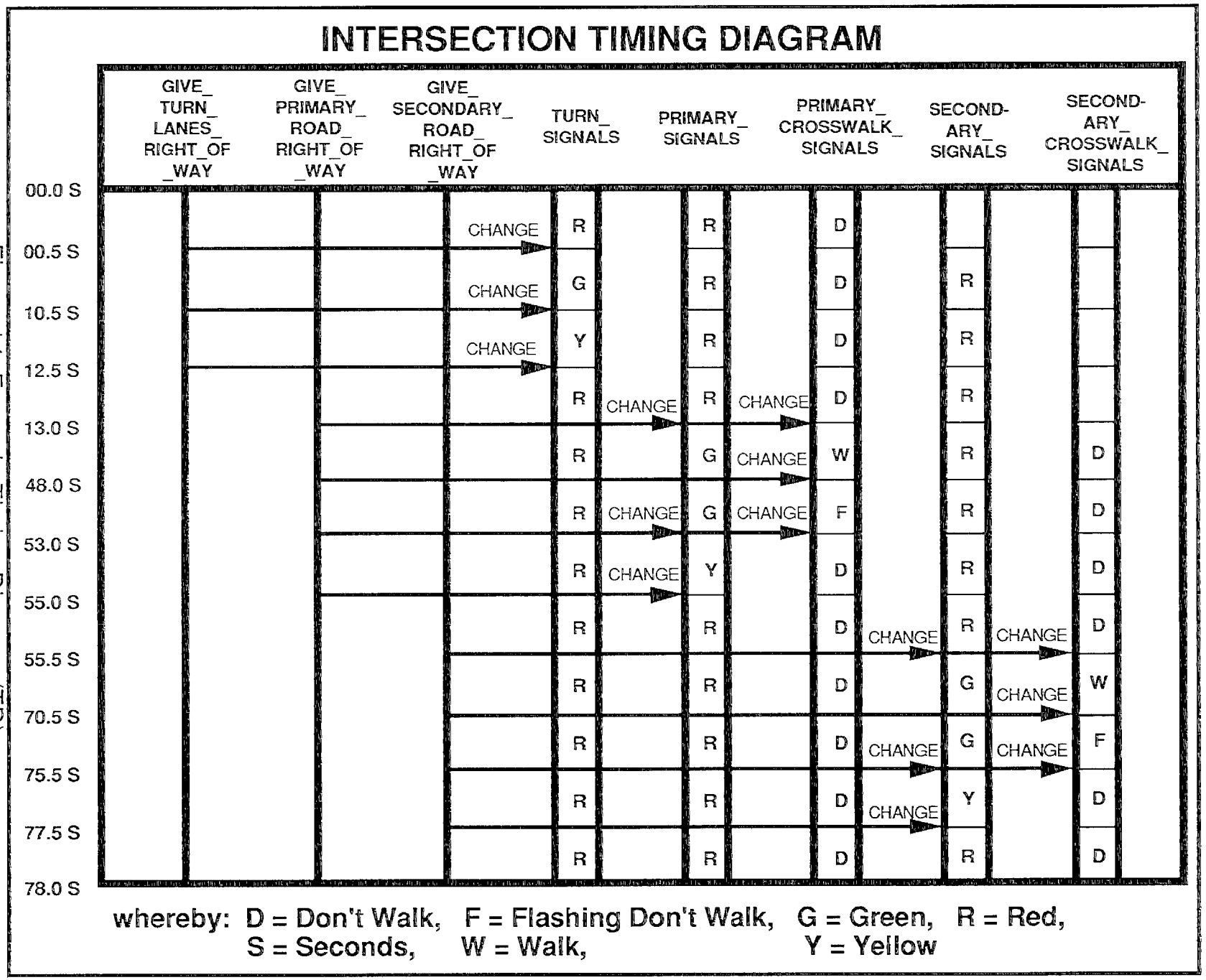

Figure 14: Example Timing Diagram (TD)