A PROPOSED INITIAL CLASSIFICATION AND EVALUATION OF ADA GRAPHIC NOTATIONS

Version 1.0 (Draft)

TO BE PRESENTED AT

TRI-Ada'90

BY

DONALD G. FIRESMITH, PRESIDENT
ADVANCED SOFTWARE TECHNOLOGY SPECIALISTS
17124 LUTZ ROAD
OSSIAN, IN 46777-9406
(219) 639-6305
(219) 747-9389 (FAX)

ABSTRACT:

This paper is divided into the four major sections. The first section describes the importance of various graphics to document and support the development of the (often object-oriented) requirements, logical design, physical design, and dynamic behavior design of Ada software. The second section describes the recommended criteria to be used to evaluate graphic and their associated notations for potential use on Ada projects. The third section presents a classification scheme that is used in the fourth section to document the major graphics currently in use in the Ada community. The fifth and concluding section presents the author's specific recommendations reguarding notation choice.

1) THE IMPORTANCE OF ADA-ORIENTED GRAPHICS

Well-written Ada is widely recognized as being highly self-documenting and is often justifiably described as being both a design and a coding language. It has numerous well-known advantages over traditional narative english Program Design Languages (PDLs), and many vendors currently provide tools that support the use of Ada or Ada PDLs directly in the design and documentation of Ada software. The Department of Defense, in violation of DODD 500.43, Acquisition Streamlining, has even mandated in DODD 3405.2, Use of Ada in Weapons Systems, the use of Ada PDLs by requiring "An Ada-based program design language (PDL) shall be used during the designing of the sofware. Use of a PDL that can be successfully compiled by a validated Ada compiler is encouraged in order to facilitate the portability of the design".

And yet, every major Ada-oriented software development method requires or suggests the use of one (or more) Ada-oriented graphics and associated notations. Some methods, such as Ed Colbert's Object-Oriented Software Development (OOSD) and Don Firesmith's Ada Development Method (ADM) include different graphics. The question is "Why?" when Ada alone should do the job.

The reasons are several. Although Ada code and PDLs are relatively self-documenting, they exist only at the extended library unit, program unit, subunit, and compilation unit level. Ada does not yet support any construct (e.g., DOD-STD-2167A Computer Software Component or CSC, subassembly, Rational subsystem) above the generic library package level. Although the relevant physical static architecture design information is implicitly in the code (e.g., via **with** clauses), the information concerning an entire CSC is scattered over numerous printouts of library units, bodies, and subunits. When a software engineer or technical manager must understand several units and their interrelationships, it is clear that "a picture is worth a thousand words" and "you can't see the forest for the trees" without the appropriate graphics. The dynamic behavior design is even less explicit because it deals with time and because subprogram and entry calls often cross extended library unit and subassembly boundries. Similarly, requirements analysis and logical design deal with different higher-level concepts than physical design, and software engineers have also noticed that the structure of the requirements and logical design need to be consistent with the structure of the physical design and code for the sake of understandability and requirements traceability. For these reasons

and others, the SIGAda Software Development Standards and Ada Working Group (SDSAWG) successfully lobbied to get the requirement to use PDLs removed from the original DOD-STD-2167.

The use of Ada-oriented graphics is therefore necessary for the proper understanding and documentation of non-trivial Ada software.

2) RECOMMENDED CRITERIA FOR GRAPHICS EVALUATION

DOCUMENT ALL ASSEMBLIES, SUBASSEMBLIES, CLASSES, AND ABSTRACT OBJECTS INCLUDING ATTRIBUTES, OPERATIONS, AND EXCEPTIONS.

FOR EACH OBJECT CLASS AND ABSTRACT OBJECT, DOCUMENT THE RELEVANT ASSOCIATED:

- REQUIREMENTS.
- LOCATION (VIA LIBRARY DIAGRAMS):
  - ASSEMBLY, SUBASSEMBLY, AND EXTENDED LIBRARY UNIT.
- IMPLEMENTATION:
  - IDENTIFIER AND VARIETY OF ADA PROGRAM UNIT OR TYPE.
  - SUBUNIT STRUCTURE (IF ANY).
- INHERITANCE RELATIONSHIPS.
- CLASSIFICATION:
  - ABSTRACT OBJECT, OBJECT CLASS, OR CLASS OF CLASSES.
  - ABSTRACT STATE MACHINE OR ABSTRACT DATA TYPE.
  - SEQUENTIAL OR CONCURRENT.
  - LIBRARY UNIT, SUBUNIT, ADA TYPE, OR ADA OBJECT.
- RESOURCES:
  - EXPORTED, HIDDEN, AND REQUIRED:
    - ATTRIBUTES (E.G., ADA DATA TYPES AND OBJECTS).
    - OPERATIONS.
    - EXCEPTIONS.
- DYNAMIC BEHAVIOR VIA:

STATE TRANSITIONS.
DATA AND CONTROL FLOWS.
CONTROL DIAGRAMS.
TIMING RELATIONSHIPS.

OBJECT-ORIENTED GRAPHICS ARE REQUIRED TO DOCUMENT **MULTIPLE** OBJECTS, CLASSES, THE ADA UNITS THAT IMPLEMENT THEM, AND THEIR RELATIONSHIPS AND INTERACTIONS.

OBJECT-ORIENTED GRAPHICS ARE REQUIRED TO DOCUMENT:

REQUIREMENTS STRUCTURE AND INTERACTIONS.
LOGICAL DESIGN.
PHYSICAL DESIGN.
STATIC ARCHITECTURE.
DYNAMIC BEHAVIOR.
EXCEPTION PROPAGATION.

GRAPHICS SHOULD BE:

UNDERSTANDABLE:
SIMPLE, CLEAR, AND INTUITIVE.
MAINTAINABLE.
ADA-ORIENTED.
OBJECT-ORIENTED.

GRAPHICS SHOULD **NOT** ENCOURAGE FUNCTIONAL APPROACHES.

BECAUSE NO SINGLE GRAPHIC CAN DOCUMENT EVERYTHING WITHOUT BECOMING CLUTTERED AND INCOMPREHENSIBLE, MULTIPLE GRAPHICS DOCUMENTING DIFFERENT ASPECTS (E.G., STATIC ARCHITECTURE, DYNAMIC BEHAVIOR) SHOULD BE USED.

GRAPHICS SHOULD NOT VIOLATE **THE MILLER (HRAIR) LIMIT** BY DOCUMENTING MORE THAN SEVEN PLUS OR MINUS TWO NODES (E.G., ABSTRACT OBJECTS OR CLASSES) ON AVERAGE.

BECAUSE BOTH OBJECTS AND CLASSES ARE COMPLETELY CHARACTERIZED FROM THE USER VIEWPOINT BY THEIR EXPORTED OPERATIONS AND EXCEPTIONS, ICONS SHOULD SHOW THESE EXPORTED RESOURCES IN ADDITION TO THE IDENTIFIER OF THE OBJECT OR CLASS.

3) ADA-ORIENTED GRAPHICS CLASSIFICATION SCHEME

Ada-oriented graphics can be classified as follows:

1) **Graphics documenting Requirements and Logical Design**

1.1) Graphics documenting Static Architecture

1.1.1) Graphics based on Object Abstraction

1.1.2) Graphics based on Semantic Relationships

1.2) Graphics documenting Dynamic Behavior

1.2.1) Graphics based on State

1.2.2) Graphics based upon Data Flow

1.2.3) Graphics based upon Timing

2) **Graphics documenting Physical Design and Code**

2.1) Graphics documenting Static Architecture

2.1.1) Graphics based upon Collections of Collections of Library Units

2.1.2) Graphics based upon Collections of Library Units

2.1.3) Graphics based on Individual Library Units

2.2) Graphics documenting Dynamic Behavior

2.2.1) Graphics based upon Calling

2.2.2) Graphics based upon Timing

4) EVALUATION OF ADA-ORIENTED GRAPHICS

DIAGRAMS BASED ON OBJECT ABSTRACTION:

ADM	SUBASSEMBLY OBJECT INTERACTION DIAGRAM (SOID)
OOSD	OBJECT INTERACTION DIAGRAM (OID)
OOSD	OBJECT HIERARCHY DIAGRAM (OHD)
OOSD	OBJECT CLASS DIAGRAM (OCD)
OOSD	WITH DEPENDENCY DIAGRAM
GOOD	OBJECT DIAGRAM

DIAGRAMS BASED ON SEMANTIC RELATIONSHIPS:

ADM	SUBASSEMBLY SEMANTIC NET (SSN)
EVB	SEMANTIC NETWORK
OOA	INFORMATION STRUCTURE DIAGRAM
	ENTITY RELATIONSHIP ATTRIBUTE (ERA) DIAGRAM

DIAGRAMS BASED ON STATE:

ADM	STATE TRANSITION DIAGRAM (STD)
OOA	STATE TRANSITION DIAGRAM (STD)
EVB	STATE TRANSITION DIAGRAM (STD)
OOSD	MEALY STATE TRANSITION DIAGRAM

DIAGRAMS BASED ON DATA FLOW:

EBDM	CLOUD DIAGRAM
ADM	OBJECT-ORIENTED DATA/CONTROL FLOW DIAGRAM (OOD/CFD)
OOA	DATA FLOW DIAGRAM (DFD)
GOOD	DATA FLOW DIAGRAM (DFD)
M-B O-O-D	DATA FLOW DIAGRAM (DFD)

DIAGRAMS BASED ON TIME:

ADM	SUBASSEMBLY TIMING DIAGRAM
BOOCH	TIMING DIAGRAM

5) CONCLUSION