N89-16373

Ada Structure Design Language
ASDL

Lutfi Chedrawi, M.S.
Applied Technology Division
System Engineering Department

Computer Sciences Corporation

Abstract

An artist acquires all the necessary tools before painting a scene. In the same analogy, a software engineer needs the necessary tools to provide his/her design with the proper means for implementation. Ada provides these tools. Yet, as an artist's painting needs a brochure to accompany it for further explanation of the scene, an Ada design also needs a document along with it to show the design in its detailed structure and hierarchial order.

Ada could be self-explanatory in small programs not exceeding fifty lines of code in length. But, in a large environment, ranging from thousands of lines and above, Ada programs need to be well documented to be preserved and maintained. The language used to specify an Ada document is called Ada Structure Design Language (ASDL). This language sets some rules to help derive a well formatted Ada detailed design document. The rules are defined to meet the needs of a project manager, a maintenance team, a programmer and a system designer. This paper will explain in detail the design document templates, the document extractor, and the rules set forth by the Ada Structure Design Language.

1.0 Introduction

Ada covers the different scopes under the software engineering spectrum. The Ada scopes can range from real time systems, scientific applications and other known software applications to abstract problems mapping, object oriented programming and new software engineering concepts.

Keeping this in mind, Ada can become very complicated when designing large projects governed by many different tasks, generic entities and overloading mechanisms. Therefore, design documents are needed to clarify some of the obscurities that might arise when designing large systems. The design document should also accomodate for the tools provided by Ada and support the Ada language by showing the program, entities, and tasks at the functional level. The design method, called Ada Structure Design Language (ASDL), approaches Ada from two different levels :

- o The specification level.
- o The functional level.

```
-- <###>

-- ***************************************************************************
-- *                    Package TASKS_INPUT_QUEUE                            *
-- ***************************************************************************

-- <###>

-- ***************************************************************************
-- *                                                                         *
-- * Author      :   Lutfi Chedrawi                                          *
-- * Company     :   CSC                                                     *
-- * Job order   :                                                           *
-- * Contract #  :                                                           *
-- *                                                                         *
-- ***************************************************************************


with UNCHECKED_DEALLOCATION;

generic
  type OBJECT is private;

package TASKS_INPUT_QUEUE is

  -- <###>
  --=========================================================================
  --                          PACKAGE OVERVIEW
  --=========================================================================
  -- Package overview ------------------------------------------------------
  -- ----------------
  -- The tasks input queue will manage the input queue for a given
  -- process. The input queue manager task in this package will either
  -- put or take an object on or off the queue repectively. This task
  -- also manages the queue by adding or releasing nodes to and from the
  -- input queue and keeping a record of the queue size.
  -------------------------------------------------------------------------
  --
  --=========================================================================
  --                          INTERNAL ROUTINES
  --=========================================================================
  -- Internal routines -- Description --------------------------------------
  -- ------------------    -----------
  -- Task
  -- QUEUE_MANAGER         To manage the queue size, the reads and wr tes
  --                       from and to the input queue. In the read
  --                       operation, if no objects were on the queue
  --                       the queue manager requester will wait until
  --                       an object is put on the queue.
  -- Procedures
  -- GET                   Performs a rendezvous with the get entry in
  --                       the queue manager task to get an object off
  --                       the queue.
  -- PUT                   Performs a rendezvous with the put entry in
  --                       the queue manager task to put an object on
  --                       the queue.
  -- QUEUE_SIZE            Performs a rendezvous with the queue size
  --                       entry in the queue manager task to get the
  --                       size of the queue in terms of number of nodes
  --                       in the queue.
  -------------------------------------------------------------------------
  --
  --=========================================================================
  --                          EXCEPTIONS
  --=========================================================================
  -- Exceptions ---------- Description -------------------------------------
  -- ----------            -----------
  -- N/A
  -------------------------------------------------------------------------
  --
```

```
--=============================================================
--                      EXTERNAL REFERENCES
--=============================================================
-- External entities -- Description ---------------------------
-- ------------------    -----------
-- N/A.
-------------------------------------------------------------
--
--=============================================================
--                      EXTERNAL ROUTINES
--=============================================================
-- External routines -- Description ---------------------------
-- ------------------    -----------
-- Procedure
-- UNCHECKED
--   _DEALLOCATION      This procedure will release the input queue
--                      nodes back to heap storage.
-------------------------------------------------------------
--
--=============================================================
--                      CHANGE HISTORY
--=============================================================
-- Date/Authors -------- Description ---------------------------
-- ------------          -----------
-- 3/31/86 - Lutfi      Added a get procedure to rendezvous with the
--                      get entry in the input queue manager task.
-- 4/1/86  - Lutfi      Added a put procedure to rendezvous with the
--                      put entry in the input queue manager task.
-- 4/1/86  - Lutfi      Added a queue size procedure to rendezvous
--                      with the queue size entry in the input queue
--                      manager task.
-------------------------------------------------------------
--=============================================================
--                      DEPENDENCY TREE
--=============================================================
-- Dependency tree ---------------------------------------------
-- ---------------
--   Task                 Calling procedures
--   QUEUE_MANAGER --
--                   |
--   Entries         |
--     o GET         |__ GET
--     o PUT         |
--     o QUEUE_SIZE  |
--                   |__ PUT
--                   |
--                   |
--                   |__ QUEUE_SIZE
--
-------------------------------------------------------------
-- (###)

-- QUEUE --> to queue the request for a task in a FIFO queue.

type QUEUE;
type QUEUE_ACCESS is access QUEUE;
type QUEUE is
  record
    FORWARD : QUEUE_ACCESS ;
    BACKWARD: QUEUE_ACCESS ;
    ELEMENT : OBJECT       ;
  end record;

-- Internal routines GET, PUT and QUEUE_SIZE
--
  procedure GET        (ELEMENT       : out OBJECT  );
  procedure PUT        (ELEMENT       : in  OBJECT  );
  procedure QUEUE_SIZE (SIZE_OF_QUEUE : out INTEGER );

end TASKS_INPUT_QUEUE;
```

Example -1- : Ada specification level design document using ASDL.

2.0 ASDL levels:

2.1 - The specification level:

ASDL, at this level, will suffice the specification definition and description of a system and put the following at hands :

- o Requirements : statement definition of the overall structure.
- o Author & History updates : a log file of updates will provide another programmer and the system manager with history information of all changes made. This pin-points the responsible person for the changes made and keep track of program progression.
  Tracking responsibilities is needed by the system manager in case any ambiguities ever arise that need further explanation or further documentation to help clarify the changes made.
- o Independability : interfaces and hierarchies definitions of each entity.
- o Maintainability : the system will be easily maintained throughout its life cycle. ASDL will provide all the clues for a maintenance team to keep track of the environment.

2.2 The functional level :

ASDL, at this level, provides programmers with tools for debugging ease and managers with prospects on design clarity . It allows, the documentation of :

- o Requirements : statement definition of an entity.
- o Structured analysis : explaining the input/output and specifications of each entity.
- o Structured design : defining the functional flow of each entity.

3.0 ASDL format

3.1 ASDL specification level format:

ASDL will show the declaration of an entity. The data structure, functions, procedures, tasks, and packages are explained at this level in a general form without going into details. The specification level design document using ASDL is shown in example -1-. A further investigation of this example allows us to identify different entries within the specification level format. Each entry permits the documentation of a part of the system that meets the needs of the different classes of people involved. All entries are mapped to a static form which allows the derivation of a specification level template. The template skeleton is static on

the outside, but the explanation within each entry can be dynamically filled with information to preserve the creator's integrity to express his own design documents.

The template entries for the specification level format serve as a road map to each or all individuals involved in the design of the entity. ASDL specification template format holds the following entries:

- o entity overview
- o internal routines
- o exceptions
- o external references
- o external routines
- o change history
- o dependency tree

The entity overview entry identifies the function of an entity. This entry serves all the classes of people involved in the development of the project. The information covered in this entry should hold the important features governining an entity. Not only would this entry serve as information coverage of the entity but also acts as a fast index to the contents of the entity under development or investigation.

The internal routines entry covers the naming definitions and the entity internal routines descriptions. A maintenance team can make use of this entry by utilizing the explanation provided to understand the problem statement definition and to identify the internal routines. Both, the project developers and system maintainers hold the responsibility of keeping the information within this entry up to date.

The exceptions, external references and external routines entries exclusively permit the system designers and project managers to recognize the system exceptions handling mechanism and to understand the system components interaction. The system designers can keep a close watch of the system by making sure that all errors are handled and a safe passage is assured by the exceptions handlers. Moreover, the exception entry will provide a fast summary of all exceptions occuring within an entity. In the same manner, the external references and routine entries will allow the project managers to check the entity interaction at both the general (External references) and specific levels (external references'internal routines.)

The history changes entry, allows the system maintainers to log all the changes made to the entity throughout its life cycle. Moreover the system developers can communicate among each other by notifying through this entry other team members of important changes.

The tree dependency nicely shows in a graphic form the entity internal hierarchy. This entry is intended to serve all the people involved in the project.

```
package body TASKS_INPUT_QUEUE is

  procedure DEALLOCATE is new UNCHEKED_DEALLOCATION (QUEUE, QUEUE_ACCESS);

  -- Input queue manager --> will retreive information from the input
  --                         queue

  task type INPUT_QUEUE_MANAGER is
    entry PUT        (ELEMENT       : in  OBJECT  );
    entry GET        (ELEMENT       : out OBJECT  );
    entry QUEUE_SIZE (SIZE_OF_QUEUE : out INTEGER );
  end;

  QUEUE_MANAGER : INPUT_QUEUE_MANAGER          ;


pragma PAGE;

-- <###>

  task body INPUT_QUEUE_MANAGER is

    -- SIZE -> to return the size of the input queue in terms of number
    --         of nodes in the queue.

    SIZE               : natural := 0;


    -- Function  : to manage the input queue. It either puts on or takes
    --             an object off the queue. The GET entry to get an object
    --             off the queue is guarded so the task will make the
    --             requestor wait until an object is put on the queue.
    --             This task will manage the queue size and return its
    --             value when requested.

    -- In        : N/A

    -- In Out    : N/A

    -- Out       : N/A


    --=================================================================
    --                              ALGORITHM
    --=================================================================
    -- Algorithm :
    --   begin
    --     loop
    --       select
    --         when size is greater than zero.
    --           accept get (element : out object) do
    --             get out object from the node at the forward link of head
    --               of the queue
    --             set the forward link of the node at the get node's backward
    --               link to the forward link of the get node.
    --             set the backward link of the node at the get node's forward
    --               link to the backward link of the get node.
    --             release get node back to heap storage
    --             decrement size
    --           end get;
    --       or
    --         accept put (element : in object) do
    --           create new node
    --           insert "in" object
    --           set new node forward link to head of the queue
    --           set new node backward link to head backward link
    --           set forward link of the node at the head backward
    --             link to new node
    --           set backward link of head to new node
    --           increment size
    --         end put;
    --       or
    --         accept queue_size (size_of_queue : out integer) do
    --           set size_of_queue to size
    --         end queue_size;
    --       end select;
    --     end loop;
    --   end input queue manager;
    --=================================================================
    -- <###>
```

```
pragma PAGE;

-- <###>

procedure GET          ( ELEMENT              : out OBJECT         ) is

-- Function  : to call the input queue manager task so it can get an
--             object off the queue.

-- In        : N/A

-- In Out    : N/A

-- Out       : element -> the object to be returned from the queue.

--------------------------------------------------------------------------
--                              ALGORITHM
--------------------------------------------------------------------------
-- Algorithm :
--   begin
--     rendezvous with the input queue manager task to get the object
--   end;
--------------------------------------------------------------------------
-- <###>

pragma PAGE;

-- <###>

procedure PUT          ( ELEMENT              : in  OBJECT         ) is

-- Function  : to call the input queue manager task so it can put an
--             object on the queue.

-- In        : element -> the object to be put on the queue.

-- In Out    : N/A

-- Out       : N/A

--------------------------------------------------------------------------
--                              ALGORITHM
--------------------------------------------------------------------------
-- Algorithm :
--   begin
--     rendezvous with the input queue manager task to put the object
--   end;
--------------------------------------------------------------------------
-- <###>

pragma PAGE;

-- <###>

procedure QUEUE_SIZE   ( SIZE_OF_QUEUE        : out INTEGER        ) is

-- Function  : to call the input queue manager task so it can get the
--             size of the queue.

-- In        : N/A

-- In Out    : N/A

-- Out       : size of queue -> the queue size to be returned from the
--                              input queue manager task.

--------------------------------------------------------------------------
--                              ALGORITHM
--------------------------------------------------------------------------
-- Algorithm :
--   begin
--     rendezvous with the input queue manager task to get the queue
--       size
--   end;
--------------------------------------------------------------------------
-- <###>

end TASKS_INPUT_QUEUE;
```

Example -2- : Ada functional level design document using ASDL.

3.2 ASDL functional level format:

ASDL will describe the system in a more detailed functional flow. ASDL will require system developers to combine Ada keywords and the english language to bring about a detailed flow of the entity, yet not cryptic to the designers or software maintainers. The system manager can also check the system logic and design structure for ambiguity, clarity, performance and possible implementation (i.e whether the entity can be implemented as described or whether the implementation is not possible due to misinterpretation of problem definition, requirements need, Ada weaknesses, etc....).

ASDL functional level format holds four entries as shown in example -2-. The "in", "in out" and "out" entries correspond to Ada parameters passing descriptions. The inclusion of these entries will entitle the system maintainers, developers, designers and managers to understand the input/output of system components. Moreover, a functional flow design is given by the algorithm entry to show the structure in its more detailed english like design.

Finally, ASDL tends to be similar to PDL (Process Design Language) at this level, which proves to be advantageous since no training is needed for individuals already familiar with PDL.

4.0 ASDL rules

4.1 ASDL specification level rules:

In general, ASDL does not impose any rigid rules. The ASDL rules for the specification level format should insure the derivation of a design document. The rules are set to give a detailed explanation of entities interactions, entity specification and data representation. The specification level format can be mapped onto the following rules :

- o new updates should be entered when necessary
- o dependency tree should be leveled to show the new entities in their hierarchial depth
- o history logs updates should cover the changes made
- o the information should be entered under the specified field to insure the extractor ability to perform its functions.

4.2 ASDL functional level rules :

At this level, ASDL requires developers to respect the outer and inner structures of Ada blocks, statements and looping mechanisms. The Ada keywords should be entered to show the Ada flow as if it was coded. Moreover, the Ada keywords should be combined with a detailed explanation in english to show the flow of Ada statements. The advantages of combining Ada keywords and english words will divulge when the implementation phase takes place. The project's implementation will become a matter of mapping the algorithm to Ada code.

5.0 ASDL extractor

5.1 The formatter extractor:

ASDL extractor is envisioned as a formatter extractor with menu driven options. When asked to format a document written by ASDL, the extractor will prompt the requestor with a menu. The menu selection can be accessed through cursor control.

ASDL formatter extractor should come with default values to allow simple extracting and echo printing of text to the specified destination file. On the other hand, if required, the extractor should perform all the necessary tasks to derive a well formatted Ada design document including centering of titles, margin justification, page numbering and other functions found in wordprocessors. Moreover, the extractor can control the part of text to be extracted from the document by a simple turn on/off flags or toggle keys (if an interactive session is requested.) Those flags are shown in example 1 & 2 as '@', '#' signs preceded and succeded by '<' and '>' designators.

6.0 Conclusion

In summary, ASDL will prolonge the software life cycle. In addition, it will allow the documentation of large systems otherwise might become very difficult to understand. Finally, ASDL will act as a communicae to all the classes of individuals involved in the system development.

---

* Ada is a registered trademark of the U.S government, Ada Joint Program office.
* ASDL extractor is still under development.

REFERENCES

[1] Barns, J. G. P. Programming In Ada. Addison-Wesley Publishing Company. Second Ed.

[2] Booch, G. Software Engineering With Ada. Benjamin/Cummings.

[3] Caine S. and Kent G. PDL - A Tool For Software Design. IEEE Computer Society Press.

[4] Linger C., Mills H. and Witt B. Structured Programming. Addison-Wesley Publishing Company.

[5] Privitera J. P. Ada Design For The Structured Design Methodology. IEEE Computer Society Press.

[6] ---, Reference Manual for the Ada programming Language. Ada Joint Program Office, Department of Defense.

[7] Zelkowitz M., Shaw A. and Gannon J. Principles Of Software Engineering Design. Prentice-Hall, Inc.