Lockheed Martin Corporation

Advanced Technology Laboratories

A & E - 2W

1 Federal Street

Camden, NJ 08102

_

Draft Build 1A

System Description

Document No. RASSP-EN-044

October 31, 1995

Table Of Contents

Appendices

Rapid Prototyping of Application-Specific Signal Processors (RASSP) Build 1ASystem Design Document

Version 1A

1. Introduction

1.1 Purpose

The intent of this document is to define the enterprise system as configured to support the benchmark 3 / UYS2 upgrade program.

1.2 Scope

1.3 Notes

The UYS2 upgrade includes several organizations (multiple locations). This build 1A system applies to the environment at the Camden NJ facility.

Note: This document is a draft / working form document.

1.4 Format

The format will be updated for consistence with standard specification practices in a later version.

2. Program Plan based on Workflows

The RASSP program is producing workflow templates, which represent various parts of sections of the RASSP methodology. The workflow expresses a series of steps (activities) of the methodology to be performed / executed in the RASSP environment. For each activity, the following information is defined: task to be performed, CAD tools required, Personnel required, and data objects needed as inputs, or produced as outputs. In addition, the precedence relationship between successive workflow steps are also performed. The workflows are executed in the enterprise environment using the DMM Workflow tool.

A project plan is a specific collection of workflows which have been customized to meet the specific requirements of the project, and the performing organization. A UYS-2 program plan, constructed from workflow segments, will be used to control project execution in the enterprise environment.

The RASSP enterprise system approach is to construct project plans utilizing workflow segments, or other project plans which are maintained in the workflow reuse system. Tools will be used in build 2 and later builds to support construction of the project plans via combination of the workflow segments.

Since the workflows also include the data object definitions, the process of combining workflows into projects also produces data object templates - from the collection of the object definitions on the individual activities. This data object template is implemented as a set of "business items" associated with the project.

2.1 Benchmark 3 Enterprise System Architecture Diagram

The enterprise system architecture diagram shown below depicts the hardware and software involved in the benchmark.

2.2 Benchmark 3 Schedule

The architecture and hardware development tasks of the UYS-2 program are planned using RASSP workflows, while the software development, system integration and test, environmental testing, and acceptance tasks are planned separately.

The benchmark 3 schedule is indicated in figure 2.2-1 (MS project gantt chart).

Figure 2.2-1 is included in this document as appendix A.

The program plan makes use of the following RASSP workflow segments:

Functional Design

Architecture Selection

Architecture Verification

ASIC (FPGA) Design

Module Preliminary Design

Module Final Design

Hierarchical Simulation

The program plan, when represented as a merge of the workflow segments is indicated in figure 2.2-2.

Figure 2-2 - Top Level Benchmark Three Workflow

Functional Design, Architecture Selection and Architecture verification are performed for the initial set of ETC algorithms. On delivery of a final set of algorithms, these process segments are performed again.

The hardware development is associated with the FPCAP and FPCTL modules, and hence there are preliminary design and final design workflow segments associated with each of these.

In addition, an FPGA workflow is included to accommodate the FPGA on the FPCAP module.

3. CAD Tool List

CAD tools which are being integrated with the enterprise system in support of the build 1A system, are indicated in the tables below. These include UNIX based tools as well as selected PC tools..

Tool summaries associated with each of the processes are indicated in the subsequent sections.

3.1 Systems Definition Tools

Systems processes are not used in the benchmark 3 program and hence are not used in the build 1A system.

3.2 Architecture Definition Tools

Architecture tools identified in the architecture workflow set are presented in the following table, The benchmark 3 column indicates if the tool is required for the build 1A system.

Architecture Tools	Benchmark 3	Status
Alta SPW		4
ANSYS	BM3	?
Ascent RDD-100	BM3	2
Aspect RRDM	BM3	4
AT&T SPEAR (SW Debugger)		2
CADRE OOA		2
Executable Specification Tool		1
ILOGIX Statemate		3
JRS architecture_definition	BM3	4
JRS assignment	BM3	4
JRS graph_development	BM3	4
JRS ProcSyn		3
JRS VHDL	BM3	4
LMMC Price-JRS		2
Marconi RTM		2
Mathworks Matlab		3
MCCI		2
MGC Autotherm	BM3	4

MGC Protoview	BM3	3
MGC VTM Top		?
MSI RAM/ILS		2
Omniview PMW		2
PGSE		?
Physical Decomposition Tool		2
QuickVHDL	BM3	4
Rome Labs TSTB/Waves		?
SDRC	BM3	3
SW Tools (misc Compilers, etc)		1
Spreadsheet	BM3	MAC/PC
Summit Visual HDL	BM3	3
TMAT Tool (SS)	BM3	MAC/PC
TSD Tool (SS)	BM3	MAC/PC
Univ. Oregon PIE (Perf. Mon.)		2
VISTA O-O VHDL (pre-proc)		2
Word Processor	BM3	MAC/PC

Status Key

1 Tool is not defined

2 Tool is not mature

3 Tool is mature and can be integrated

4 Tool has already been integrated

5 Tool has been integrated for BM3

? Status of tool is unknown

MAC/PC is a MAC or PC tool

3.3 Detailed Design Tools

The design tools associated with detailed design which are integrated with the enterprise system are indicated below. Tools associated with subsequent builds are also indicated.. Tools indicated by BM3 are actually being used on the benchmark program, and hence are included in build 1A system.

Detailed Design Tools	BM3?	Status	FPGA Design	Module Design	Module Final Design
Aspect RRDM	?	4	BM3	BM3	BM3
IKOS Voyager FS	?	2			
Interleaf TPS	?	3			
LMG HW Modeler	TBD	3			
LMG Smartmodels	?	3		BM3	
LogicVision ICBIST		2			
MGC AutoTherm	?	3		BM3	BM3
MGC BoardStation	?	4		BM3	BM3
MGC DA-LMS	?	4		BM3	
MGC FabLink	?	3			
MGC PTM Site	?	?			BM3
MGC QuickPath		3		BM3	BM3
MGC QuickVHDL	?	3	BM3	BM3	BM3
MGC QvPro		3		BM3	
MGC VHDLWrite	?	3		BM3	
MGC VTM:TOP	?	3		BM3	
NeoCAD	?	3	BM3		
Omniview Fidelity		2,3			
Precedence Sim. Backplane		2,3			
Quickturn Emulator		3			
SCRA AP210 Translator		4			
SCRA Producibility Advisor		4			

SPICE (analog sim)	?	3			
Summit TDS	?	3			BM3
Summit VisualHDL	?	4		BM3	
Synopsys Design Compiler	?	3	BM3		
Teradyne Victory	?	3			
TI Asset	?	PC			
TSTB/WAVES	?	?	BM3		

Status Key

1 Tool is not defined

2 Tool is not mature

3 Tool is mature and can be integrated

4 Tool has already been integrated

5 Tool has been integrated for BM3

? Status of tool is unknown

MAC/PC is a MAC or PC tool

4. Workflows

Workflow Modeling Overview / General Cases

4.1 Detailed Workflows (IDEF3X) - The workflows associated with build 1A system are as follows:

Functional Design

Architecture Selection

Architecture Verification

ASIC (FPGA) Design

Module Preliminary Design

Module Final Design

Hierarchical Simulation

The architecture workflows are included twice in the program plan - once for ETC Initial, and once for ETC final.

Likewise Module Preliminary Design and Module Final design are repeated for each of the two modules (FPCTL, and FPCAP).

Detailed definition of the workflows are included in Appendix A of the methodology document.

4.2 Business Item Definitions

Business items represent collections of information to be managed in the enterprise system assembled under a logical identifier (business item name). Business items also have states associated with them. The states defined include:

Business items will be managed in the enterprise data management system during execution of the project. Business items will be either copied out or checked out of the data management system on execution of particular workflow steps, and returned (updated) to the data management system on completion of the steps - the specific approaches for handling the business items on task start and task completion are included with the definition of each step. In the simplest case, a business item is checked out at the start of a task, and returned on completion of the task.

A set of business items are defined initially for a project - each workflow segment identifies a set of business items that are utilized in execution of the workflow segment. The workflow segments are instantiated for the particular application task on the project. For example: The library workflow associated with module preliminary design is instantiated for the FPCAP module, and for the FPCTL module. In instantiating the workflow segment, both the tasks and business items are instantiated for each specific case of the workflow.

Business Item State Definitions

Preprocessing Requirements

Postprocessing Requirements

4.3 Logical Referencing for Tools

4.4 Special Concurrent Engineering Cases / Workflow Approaches

4.4.1 Multiple Concurrent Tasks using Parallel Copies of Data Sets

4.4.2 Multiple Design Alternatives Developed Concurrently

4.4.3 Pipeline of Data Sets Thru a Workflow

4.4.4 Failback Paths

4.4.5 Multiple Iterations in a Workflow

5. RASSP Data Management System (DMS)

The RASSP data management system is responsible for supplying the correct data set for each process step. This function is transparently performed in conjunction with the workflow manager. The data manager also provides the capability to access data independent of the process step or workflow. Users have access to the data manager interface to perform any functions necessary to get the job done.

5.1 Overview of Functionality

The DMS will provide the ability to define users and their access authorizations, define workspaces and configuration management, business items that flow through the processes and their corresponding data items, and the ability to manage all the information related to a project as a data set. The DM2.0 core functionality will be tailored to support the RASSP Benchmark 3 processes.

5.1b Overview of the Information Model

The following diagram shows a conceptual information model to support the project management concepts of the RASSP enterprise system. The model shows how information about projects, workflows, process steps, and business items, etc. relate to each other.

5.2 Configuration Management

Configuration management is addressed with a hierarchy of workspaces - implemented in the data management system. Shared workspaces will be defined for the project - the business items will be created in the shared workspace at the start of the project (a product of the project creation phase).

5.2.1. Workspaces

5.2.1.1 Shared and Private Workspaces

Workspaces **are partitions of the design object space to allow designers working on the various parts of a project to selectively make their design objects visible to others in the project [Cattell, 1991]. In the RASSP Configuration Management (CM) model, three types of workspaces exist: private, shared, and global. Workspaces are organized in a hierarchical fashion as shown in Figure 1.1. Each node in the hierarchy represents a workspace. Branches in the hierarchy represent a parent-child relationship between workspaces. The** ***global workspace*** **is at the root of the hierarchy,** ***shared workspaces*** **are the intermediate nodes in the hierarchy, and** ***private workspaces*** **are the leaves in the hierarchy. [Martin Marietta, 1994]**

5.2.1.2 Workspaces in DM 2.0

The above workspace hierarchy can be implemented in DM 2.0 using the features of *users* and *vaults*. Relationships between workspaces may be enforced by defining *groups*, which contain related users, and limiting the access of these groups through the use of *rules*.

In DM 2.0, each user has a private workspace. That is, there is a one-to-one mapping between a user and a private workspace. Rules may be used to enforce the privacy of individual workspaces. Shared workspaces can be implemented through the use of vaults. A vault is a logical collection of shared objects. Rules can be used to control access to a vault. User-to-vault relationships can be established to allow visibility of ancestor workspaces. The global workspace will consist of selected data from all shared workspaces (vaults), obtained through the DM 2.0 *query* capability.

The proposed DM 2.0 implementation of the RASSP workspace hierarchy is shown in Figure 1.2.

***** Legend:

= Represents the physical file system location for the items residing in the associated vault (shared workspace) or user (private workspace). Vault and work locations may be on the same host machine or on separate machines within a network.

VL = Vault Location: Each vault location will be associated with only 1 vault.

WL = Work Location: Each work location will be associated with only 1 user.

= Represents a DM 2.0 saved query.

SQ = Saved Query: Every user will have access to the Global Workspace through a predefined query.

= Represents a logical collection of objects based on ownership. An owner of an object is either a user or a vault.

Each vault will have 1 or more vault locations associated with it.

Each user may be associated with 1 or more vaults.

Each user will have 1 or more work locations associated with it.

The version numbers used in the above figure do not reflect the DM 2.0 naming convention, but are used to provide clarity to the figure.

5.2.2. Data Object Management

5.2.2.1 Configurations and Version Control

The RASSP CM Model uses a data object versioning scheme where related data objects that evolve at the same time are grouped together as a configuration. At any point in its life cycle, a configuration can exists in one of three states: transient, working, or released. Upon creation, a configuration is considered to be transient and is associated with a private workspace. A transient version of a configuration allows for updates and may be deleted. A transient version may be promoted to a working version when the configuration has reached a level of maturity such that it can be shared with other users. Working versions reside in shared workspaces. At this state the configuration cannot be updated, but may be deleted. A configuration is considered to be in the released state when a working version of that configuration is promoted to the global workspace. Released configurations cannot be updated or deleted. [Martin Marietta, 1994]

A transient version of a configuration may be created from a previous version regardless of its state. The source configuration remains unchanged if it is a working or released version. Creation of a transient version from already existing transient configuration causes the source configuration to be promoted to the working version level. A configuration may be deleted if it is at the transient or working version level and is at the lowest level in a workspace hierarchy. [Martin Marietta, 1994]

5.2.2.2 DM 2.0 Strategy For Version Control

The RASSP scheme for version control can be implemented using DM 2.0 features. DM 2.0 is capable of managing individual objects as well as collections of objects. Hence, a configuration as described in Section 2.1 may be a group of one to many objects. For the purpose documenting an implementation of the RASSP CM Model in DM 2.0 a folder will be used as an example of a configuration. A folder is a mechanism for grouping a set of objects together for a specific purpose. The objects may be of different classes and may be added or removed as required.

A folder will be considered transient if it is created by a user. At this point, the folder may be updated or deleted. A transient folder may be created in one of the following ways--

- initially, by the folder option in the DM create menu in a userís workspace
- a copy action on an existing folder, resulting in a newly named folder, regardless of the state that the source folder is in
- a check out action on an existing working folder in a vault
- a revise on a released folder in a vault.

Transferring ownership of a folder from an individual user to a vault ìpromotesî the folder to a working version. A working version of a folder may be checked out, baselined, or deleted. A released version of a folder will result from baselining a working version. A released folder may not be updated or deleted. Revising a folder generates a new copy of the folder which can then be manipulated. Therefore, the global workspace will consist of all baselined objects (including folders) and may be accessed by a predefined query. All users will have the ability to exercise this query.

Figure 2.1 graphically depicts the DM 2.0 version scheme.

5.2.3. CM Functions

5.2.3.1 Workspace Functions

A *global workspace* in DM 2.0 is mapped to all baselined objects in all vaults within a database. Items are visible in a global workspace by use of the DM 2.0 Saved Query Object Class.

A *shared workspace* in DM 2.0 is mapped to a *vault/vault location* and can be accessed by performing transfer, check in, and check out operations. A vault is a logical collection of shared objects. A vault may contain data objects or actual file system items. A vault location provides a file system location for storing physical files owned by the vault.

In DM 2.0, each user has a *private workspace.* Based on projects, a parent-child relationship can be established between private and shared workspaces. A private workspace may have more than one *work location.* Similar to vault locations, a work location provides actual file system space for objects residing in a private workspace. Note: A private workspace may have relationships with more than one shared workspace.

5.2.3.1.1 Creating a workspace

In DM 2.0, a workspace is created by default upon creation of a user. Shared workspaces are created upon creation of a vault/vault location. Rules dictate parent-child relationship between private and shared workspaces. The global workspace is composed of all baselined objects within all vaults contained in the database. The following steps/concepts serve as a guide in developing the DM 2.0 implementation of workspaces within the RASSP CM model--

- administration authority is required to create users and vaults (i.e., private workspaces and shared workspaces)
- the global workspace is dynamic in nature and evolves as objects are baselined in project vaults
- vaults will be created with the *noreplace* attribute set to prevent the overwriting of shared data items
- work locations must be created by workspace owners due to write privileges enforced by the operating system.

5.2.3.1.2 Accessing an arbitrary workspace

In DM 2.0, the ability to access an arbitrary workspace is controlled by the underlying rules in relation to the requester. Accessing an arbitrary workspace could involve no more than performing a query on that workspace to see what items exist in that workspace. It may also involve copying, updating, transferring, check in/out, baselining, or revising items. Each action is controlled by rules in relation to project, vault, role, and group assignments. The following steps/concepts will guide the DM 2.0 implementation of accessing workspaces within the RASSP CM model--

- super users will have access to all workspaces within the system
- access to shared workspaces (vault/vault locations) will be restricted based on need (i.e., need to query an item, need to update an item, need to baseline, or revise an item)
- access to private workspaces will be restricted by the rules that control the actions available to non-owners
- all users will have access to the global workspace (defined within the context of a database).

5.2.3.1.3 Accessing child workspaces

In DM 2.0, the workspace hierarchy is implemented as relationships defined between an individual userís private workspace and that userís ability to access shared workspaces (vault/vault locations). Shared workspaces will be allowed as parents of private workspaces, and can have more than one child workspace (that is, more than one user) at a time. The ability to access child workspaces will be provided through the DM 2.0 query method, which will return a list of users who have access to a shared workspace. This data is for information only since a private workspace cannot be a parent and a shared workspace will not have more than query access to a private workspace.

5.2.3.1.4 Accessing the parent workspace

The DM 2.0 implementation for accessing a parent workspace is much the same as accessing an arbitrary workspace. Rules will limit the scope of access to ancestor workspaces residing between the current workspace and the global workspace. The following DM 2.0 mechanisms will be available to use when accessing parent workspaces: query, copy, update, transfer, check in/out, baseline, and revise. Actions will be controlled in relation to project, vault, role, and group assignments. The following steps/concepts will guide the DM 2.0 implementation of accessing a parent workspace within the RASSP CM model--

- super users will have access to all workspaces within the system
- access will be restricted based on the relationship between the current workspace and the desired parent workspace
- all users will have access to the global workspace (defined within the context of a database).

5.2.3.1.5 Making a workspace visible

The RASSP CM model calls for the ability to link an application to a specified workspace. The application would then have access to all items in that workspace and its ancestor workspaces. This can be accomplished through tool encapsulation within DM 2.0. Once a tool (application) has been encapsulated, it may be launched via the graphical user interface (GUI) double-click or drag-and-drop capabilities. Rules will govern which users have access to certain applications. Once the application is active within the context of a workspace, the application may manipulate all associated data items in the current workspace and its ancestor workspaces.

For example: An AutoCAD application would be able to manipulate Drawing files residing in work and vault locations associated with the current private workspace and its associated shared workspaces.

5.2.3.2 Version Management Functions

The RASSP concept of a configuration will be implemented in DM 2.0 as a folder.

5.2.3.2.1 Creating a configuration

In DM 2.0, creation of a folder is achieved through the point-and-click capability of the GUI. When created, the state of a folder is transient (i.e., it will reside in a userís private workspace). The following steps/concepts will guide the DM 2.0 implementation of creating a configuration within the RASSP CM model--

- super users will be allowed to create folders in any available private workspace
- administrators will have authority to create folders in the private workspaces for which rules allow

them access based on established project-to-user relationships
- userís will be restricted to folder creation within their private workspace areas only.

5.2.3.2.2 Inserting data objects into a configuration

Once a folder is created in DM 2.0, items may be inserted into it through the use of the drag-and-drop capability of the GUI. Updates will be allowed to transient level folders only. The following steps/concepts will guide the DM 2.0 implementation of insertion into configurations within the RASSP CM model--

- super users will be allowed to insert data into a folder regardless of its associated private workspace location
- administrators will be allowed to update folders residing in private workspaces for which rules allow them access
- users will be restricted to updating folders within their private workspace areas only
- working and released folders may not be updated.

5.2.3.2.3 Check out

DM 2.0 provides a *check out* option that creates a new copy of the selected working folder, and allows the new copy to be modified. The original folder is superseded. File system items attached to the folder being checked out are copied to the current workspace with the same relative location as the original folder. A transient folder must first be checked in before it may be checked out. Thus a folder will be at the working version level prior to check out. The following steps/concepts will guide the DM 2.0 implementation of checking out a configuration within the RASSP CM model--

- super users will be allowed to check out any folder from the shared workspace it resides in
- other users will be allowed to check out folders residing in shared workspaces for which rules allow them access

baselined folders may not be checked out.

5.2.3.2.4 Check in

DM 2.0 contains a check in feature which allows a folder to be returned to a shared workspace (vault), or transferred to a vault for the first time. If transferred

for the first time, the folder becomes visible to the shared workspace and all itís children. A check in will not replace a predecessor of the same folder. Once a folder is at a mature state and is ready to be released, it may be ìpromotedî to the global workspace by using the DM 2.0 baseline feature. All items attached to the folder are baselined as well. Changes are made to a released folder through the use of the revise feature. This feature creates the next revision of the released folder. The following steps/concepts will guide the DM 2.0 implementation of checking in a configuration within the RASSP CM model--

- super users will have the ability to check in, baseline, and revise folders regardless of that folderís private/shared workspace location
- administrators will be allowed to check in, baseline, and revise folders residing in shared workspaces for which rules allow them access
- users will be allowed to check in/transfer ownership of a folder to shared workspaces for which rules allow them access
- baseline will work only on folders which are at the working version level
- revise will work only on released folders.

5.2.3.2.5 Accessing child versions

The DM 2.0 expand relationship feature may be used to show a folderís relationships to other objects. DM 2.0 allows relationships between objects to be displayed in a query window by selecting a folder and choosing from the available options under the info menu. If desired, DM 2.0 can also provide a ìtreeî like view of these relationships. Child versions of a folder could be obtained by expanding the ìis superseded byî relationship. The following steps/concepts will guide the DM 2.0 implementation of accessing child versions of a configuration within the RASSP CM model--

- super users will have the ability to examine the relationships of any folder residing in any workspace location
- administrators and users will be allowed to examine relationships of folders residing in workspaces for which rules allow them access.

5.2.3.2.6 Accessing parent versions

The DM 2.0 expand relationship feature may be used to show a folderís relationships to other objects. DM 2.0 allows relationships between objects to be displayed in a query window by selecting a folder and choosing from the available options under the info menu. If desired, DM 2.0 can also provide a ìtreeî like view of these relationships. Parent versions of a folder could be obtained by expanding the ìsupersedesî relationship. The following steps/concepts will guide the DM 2.0 implementation of accessing parent versions of a configuration within the RASSP CM model--

- super users will have the ability to examine the relationships of any folder residing in any workspace location
- administrators and users will be allowed to examine relationships of folders residing in workspaces for which rules allow them access.

5.2.3.2.7 Naming versions

DM 2.0 will not allow the name attribute of a folder to be directly changed. Renaming a folder can be indirectly accomplished by using the DM 2.0 copy feature. Copying the folder will produce a new folder object containing the same attributes as the source folder, but with a different name. The following steps/concepts will guide the DM 2.0 implementation of naming versions of a configuration within the RASSP CM model--

- super users will have the ability to copy any folder
- administrators and users will have copy privileges for only those folders residing in workspaces for which rules allow them access.

5.2.3.2.8 Retrieving a named version

The DM 2.0 query feature may be used to access the named version of a given folder based upon attribute values in the search criteria. The following steps/concepts will guide the DM 2.0 implementation of retrieving named versions of a configuration within the RASSP CM model--

- super users will have the ability to query for folders residing in any workspace location
- administrators and users will be allowed to query for folders residing in workspaces for which rules allow them access.

5.3 Build 1A User Definitions

Users of the build 1A system are classified into the following categories (determines authorization privileges in the system:

DM2Admin. This person is the super user of the Build 1A / Metaphase system. He can do everything plus anything that the following users can do.

RASSP Administrator. This person is the all powerful user with in the RASSP system and CM model. He can do everything the following users can do. The difference in this user and the DM2Admin person is that the RASSP Administrator cannot alter the rules governing the RASSP CM model, etc.

User Manager. Sole responsibility is to create users in the system, assign them to user group, and perform user functions when a user is not present.

Project Manager. Has control over the project shared workspace and is a member of the project team. This person also assigns users to the project that he is in charge of.

Project member. A member of the project. Can put things in the project shared workspace. In charge of his own items within the context of a project.

User. Every user should be in this group.

5.3 Project Workspace Definitions

5.4 Other Data Management Enhancements

5.4.1 DM M / DM2.0 Interaction

5.5 Business Item Definitions

6. DMM / Project Mgmt Capabilities

The project plan is implemented as a single workflow in Design Methodology Manager.

Modifications to the project plan (workflow) is an administrator function - user interaction will be through the DMM Displayer.

6.1 Execution Control

Each process step as specified in the IDEF model for the workflow will be executed by a DMM process script which is invoked when the user "clicks" on the appropriate step in the process.

6.2 Tool Launch Pad Operation

6.3 Data Management Interaction

6.4 Status Logging Capability

DMM Toolpad will maintain a status log of events occurring in the execution of the project plan. This status log viewable from the DM User interface, A file containing the status log is also exportable in a CSV format - compatible with Microsoft Project.

The following are metrics are to be supported - the approach for support with DMM / Toolpad is indicated in Italics

1. Number of Iterations Thru a Path / Step - For a process step that is executed multiple times (feedback path) - desire to know the status each time the step is executed.

DMM currently keeps a status log which has an entry for each process step that is used: Step #, User, Time Stamp, Status (started / finish / fail / etc.). An entry is made for each event that happens with a process step.

2. Tool Usage - Desire to know the elapsed time for tool usage per process step.

Because we are using a tool launch pad (multiple tools per step) - the DMM status log does not track the actual tool usage. An additional status log will be generated by the tool launch pad - which will include the tool usage (launch and exit time stamps)

3. Person Per Step - Desire to track the person executing each process step.

This is currently tracked in the DMM status log.

4. Notes Per Process Step - Desire to have a form to be filled out on completion of process steps by the engineer on completion of the process step - information needs to be kept with status log.

The completion status dialog box that is used at completion of a process step includes a comment field - which can be used for this purpose. This information is maintained with the status log entry.

6.5 Interface with MS Project

A list of the tasks associated with the project is exported from DMM as a CSV file - which can be imported to MS Project. Editing of the file may be required to resolve consistency issues with the task definitions in DMM and MSProject (support for hierarchy in the interface is the issue).

7. Build 1A Hardware Configuration

The following represents the hardware complement defined for the build 1A system. Systems without a userid indicated are physically located in the RASSP design center; where a userid in indicated - the location is the users desktop.

ARIEL (Sparc 10) (SunOS) 166.20.232.173 (jmeckley)

BISHOP (Sparc 20) (SunOS) 166.120. (License Server)

CYGNUS (Sparc 10) (SunOS) 166.20.232.121 (cfry)

DONALD (Sparc 10) (SunOS) 166.20.232.126 (lkline)

DORADO (Sparc 10) (SunOS) 166.20.232.124 (cstrasbe)

HICKS (Sparc 20) (SunOS) 166.120.

HUDSON (Sparc 20) (SunOS) 166.120.

MERCURY (Sparc 10) (SunOS) 166.20.232.120 (jpridgen)

MICKY (Sparc 10) (SunOS) 166.20.232.125 (hzebrowi)

MOLLY (Sparc 10) (SunOS) 166.20.236.252 (ssharma)

NEWT (Sparc 5) (SunOS) 166.120.

PAVO (Sparc 10) (SunOS) 166.20.232.123 (chein)

RIPLEY (Sparc 5) (SunOS)

SATURN (Sparc 10) (SunOS) 166.20.232.122 (rjaffe)

SULACO (Sparc 5) (Solaris) (File Server)

Intergraph TD2 (RASSP Lab)

8. Reuse Library System Definition - RASSP Reuse Data Manager (RRDM)

8.1 Aspect Explore-CIS Software Baseline

The Build 1A Benchmark 3 RRDM is implemented in Aspect's Explore-CISVersion 2.4.0 system. This release is supported by a Solaris 2.4database and server environment, built on Oracle 7.0.16.4 for Solaris, with SunOS and Solaris client applications. It also includes Aspect's read-only API version 1.0.0 for SunOS and Solaris, which may be used for integration purposes, as needed.

8.2 ATL User Environment

The software is installed on sulaco, with client access through the standard ATL environment. To run it from any SunOS or Solaris workstation on the ATL/RASSP network, execute the following commands:

```
% cd /opt/aspect

% source .cshrc

% cd bin

% startObjectViewer
```

ATL Benchmark team members will have their own accounts, and initially may access the system by entering their ATL UNIX account name for both the username and password. Passwords may then be changed via the user interface, as desired.

8.3 Benchmark Reuse/Reference Database

The initial Benchmark 3 implementation includes the Aspect VIP 1/2 Reference Database, which contains component information for ~500K integrated circuit, discrete, and passive component classes, with online databook information available for all parts. Online data dictionaries are also associated with each class of components, and may be viewed as attachments for any component record.

In addition to the Aspect reference information, an initial class hierarchy has been implemented for Assembly (Module and Substrate)and System classes, as well as for a variety of hardware design views(logic symbols, geometries, LMS catalog information, etc.) Although manufacturer reference data for the assembly and system classes is limited, work is ongoing to load all of the EPI released Mentor Graphics library data and associate that data with the related manufacturer component classes. This effort should be complete by the end of October, 1995.

Architecture Design classes, based on the RASSP Model Year Architecture report, have also been implemented in this first release of the RASSP Reuse Design Object Classification Hierarchy (RDOCH). A number of these classes have been populated with the available architecture information from the JRS NetSyn environment.

Algorithm Design classes, defined as per the MCCI Specification, Q003 Specification, Alta Group documentation, and JRS NetSyn documentation, are also included in this release. Data dictionaries and actual population of the ddatabase will be completed over the next 2-3 months.

A number of Software Design classes have been identified for implementation based on information available through the Reuse Library Library Interoperability Group (RIG), various government organizations (NASA, ARPA, NSA, etc.), and internal Lockheed corporate software libraries. These classes will be implemented and populated after the first of the year, as a part of ongoing support for the program.

Design View classes, which apply to all of the four major reuse areas identified to date, include Design For Test, Simulation Model, and Documentation classes. The Simulation Model class definition is complete, and is based on the taxonomy presented at the '95 RASSP Annual Conference by Carl Hein. A few program-related models have been classified as examples from benchmark 2, and additional models and libraries of models will be loaded as a part of the ongoing maintenance effort. Documentation classes include all of the VIP 1.2 electronic data books, as well as product information that has been provided by some of the tool vendors on the program. These classes will be continually updated as more documentation becomes available.

Supplier/Manufacturer classes have been implemented based on the GE Consolidated Purchasing System (CPS) definition. This system, which is currently in use at ATL, supports supplier management and procurement, and has been partially implemented in the RRDM for demonstration purposes as well as to decrease the procurement cycle for the program, where possible.

A couple of the RASSP Product & Services classes have been defined at this point in the program, including Design Tool and Specifications & Standards classes. The design tool class is currently under development, and will include configuration information and online documentation for all design tools encapsulated in the RASSP environment, as available. Specs and standards used on the program will be sourced from the appropriate organizations and included, again, as they become available.

8. Manufacturing Interface

9. Notes

10. Miscellaneous

10.1 Glossary

10.2 References

Appendices

This section contains specific note on implementation aspects of the build 1A system. This section is included for documentation purposes - regular updates are expected to be required.

A.1 Enterprise Startup Procedure

The startup of the Enterprise Framework requires actions to be performed by a System Administrator as well as actions performed by each user. These actions are outlined as follows:

System Administrator actions:

1) Start Oracle database on bishop as follows:

a) Login user: oradm

password: rassp

b) Type bin/dbstart

2) Start DMM Server on bishop

a) Verify the mce daemon is running on bishop.

This should start at boot time. If it isn't,

it must be start as root on bishop. Login

to bishop. Change directory to

/opt/ingr/vbest14/exec/vbdmm. Type rc.dmmbk

3) Start Metaphase server daemons on bishop:

a) Login user dm20adm

password dm20adm

b) Type muxstart, then type dspstart

4) Start the Metaphase client daemons.

a) Repeat step 3 on client machines.

5) Start the Enterprise Desktop on client.

a) At users login prompt, type projlist&.

A.2 Naming Conventions used for Build 1A Environment / Project

The following naming conventions are being used within the build 1A system - information included here for diagnistic purposes.

environment variables (defined in env):

RASSP_PROJ_VL_HOST - the host name where the project's vault locations will be

RASSP_PROJ_VL_ROOT - the full path to the dir where the project's vault locations will be

CURRENT_PROJ_NAME - current project name

CURRENT_PROJ_DIR - current project dir (full path)

Project name: PROJ_NAME

A string of upto 8 chars

Project dir name:

$PROJ_NAME.prj

Process number: PROC_NUMBER

digits separated by dots (e.g. 2.3.1.12)

Process name:

A string with first word being $PROC_NUMBER
(e.g. 2.3.1.12 Schematic Editing)
Process id: PROC_ID

proc$PROC_NUMBER

Global Vault:

Global Vault

Global Vault Location name:

Global VL

Project Vault: PROJ_VAULT

$PROJ_NAME's Vault

Project Shared Vault Location name:

$PROJ_NAME's Shared VL

Project shared vault location dir:

$RASSP_PROJ_VL_HOST:$RASSP_PROJ_VL_ROOT/$PROJ_NAME/Shared.vl

Project process' vault location name:

$PROJ_NAME's $PROC_ID VL

Project process' vault location dir:

$RASSP_PROJ_VL_HOST:$RASSP_PROJ_VL_ROOT/$PROJ_NAME/$PROC_ID.vl

Project's Shared user:

$PROJ_NAME

Project's Shared Work Location:

$PROJ_NAME's WL

$RASSP_PROJ_VL_HOST:$RASSP_PROJ_VL_ROOT/$PROJ_NAME/Shared.wl

User's Private Work Location:

$USER's WL

$HOME/$USER.wl

A.3 DMM Process Script

The approach for encapsulation is to encode tool and data item lists required for particular process steps as attributes for the process steps. A script generation routine is used to generate the actual process script for execution, on invocation of the process step.

The following is the proposed sequence of activities to be performed

by a DMM process script - in controlling the task execution of a process step..

(Please comment and answer the questions if you know them.).

New process script structure:

Step 1. Initial setup

setup necessary environment variables

setup necessary local variables

Step 2. Input data gathering

for input data, either copy or checkout them as specified in
the process attributes. if error then exit script. if an input
is to be modified to become output, then the input will always
be checked-out

for control data, either reference or copy them as specified
in the process attributes. if error then exit script

for output data, if it is to be created from scratch, then checkout
the place holder from the shared vault (the place holder is
assumed in the shared vault). if the output is to be modified from
an input data, then do nothing, because the input should have been
checked-out (see above)

Step 3. Tool launch setup

for tools, present them to the user in toolpad. the data for
each tool is set according to the process attributes

if user choose to transit the process from the toolpad, then
continue the script, else if user click Cancel from the toolpad
then exit the script

Step 4. Output data verification

if a process finished successfully, the output data should be
present in proper location. if it is modified from an input,

then checkin it. if it is created from scratch, then the user
is responsible to attach the physical data to the place holder.
the script will checkin it.

if error then exit script

Step 5. Input data restoring

for input data, dispose them as specified in the process attributes.
if an input is modified to an output, it should have been checked-in
(see above 4.). if error then exit script

for control data, dispose them as specified in the process attributes.
if error then exit script

Step 6. DMM notification

present to the user a list of exit statuses, notify the DMM
with the status the user picked.

Script flow control for the automatically generated DMM process script:

Concept:
when a user double click at a process in the DMM
Displayer, a script is generated based on the attributed defined
for the process. During the execution of the script, an error
might occur due to unusual situation. For example, the script
successfully finished Steps 1 - 4, but failed at Step 5,

this resulted in an error. Whenever an error is
encountered, the script will stop the execution and exit.

Now, say, the user solved the problem that caused the fail of Step 5, and double clicked at the process in the DMM Displayer again. Without the flow control, the same script would then be generated for the process and executed, the output would be checked-out, and then checked-in even though it had been done before.

With the Script Flow Control, however, depending on how the script was executed in the previous run, a script can be generated such that some of the Steps is skipped. In the above example, the checkout and checkin of the output data is skipped.

Implemented flow control features:

1. If Step 4 finished successfully, but Step 5 failed, then when the next the script is generated:

if the output data is created from scratch, the output is not checked-out in Step 1

if the output data is modified from an input data, the input is not checked-out in Step 1

the output data is not checked-in in Step 4

2. If Step 5 finished successfully, but Step 6 failed, then when the

next the script is generated:

Steps 1 - 5 are skipped

A.4 Automatic Script Generation Approach

A list of attributes will be attached to a process in a DMM workflow. A script generating application will use these attributes to create a script and then run the script.

Attributes

An attribute named DataList will be used to specify a list of data

items involved in a workflow process. DataList's value is a list of

(data) attribute names separated by colons:

DataList Data1:Data2:...:DataN[:]

where Data1, Data2, ..., DataN are attribute names to be defined in

separate attribute line. They will be called data attribute in this

note.

Each data attribute's value contain a list of fields separated by

colons. These fields are defined as follows:

DataI

Data Name:Data State:Data Type[:[Destination[:[Retrieve[:[Dispose[:]]]]]]

This is in BNF format

where

Data Name (Required) name of the data (IDEF3 data name)

May contain spaces between words.

Data State state of the data (IDEF3 data state)

May contain spaces between words.

Data Type (Required) Input, Output, or Control

(IDEF3 term)

Destination (Optional) Private or SharedWL or SharedVL

or Global or "",

"" means not defined, use default

For input, it specifies where the data is

checked-out/copied to. The valid values are

Private or SharedWL. The default is the

Private.

For output, it specifies where the data is

to be created. The valid values are

Private or SharedWL. The default is Private.

For control, if it is to be checked-out/

copied, it specifies where the data is

checked-out/copied to. The valid values are

Private or SharedWL. The default is
Private. If it is to be referenced,
it specifies where the data is referred
from. The valid values are SharedWL or
SharedVL or Global. The default is SharedVL.

Retrieve (Optional) how data is retrieved:
copy copy from shared vault
checkout check out from shared vault
reference reference from a location
"" not defined, use default

The default Retrieve is based on the
Data Type:

Input copy
Control reference
Output checkout

Dispose (Optional) how data is disposed:
delcopy delete the copy
delete delete the data
checkin check in to shared vault
nochange leave the data alone
"" not defined, use default

The default Dispose is based on the
Data's Retrieve value:

copy delcopy

checkout checkin

reference nochange

Note: Retrieve-Dispose compatability:

if a data is copied out, it can be delcopied by using the

document name, or deleted by using the full path,

or nochange

if a data is checked-out, it can be deleted, or checked-in,

or nochange

if a data is referenced, no dispose is necessary, hence

using nochange

An attribute named ToolList will be used to specify a list of tool items involved in a workflow process. ToolList's value is a list of (tool) attribute names separated by colons:

ToolList Tool1:Tool2:...:ToolN[:]

where Tool1, Tool2, ..., ToolN are attribute names to be defined in separate attribute line. They will be called tool attribute in this note.

Each tool attribute's value contain a list of fields separated by colons. These fields are defined as follows:

ToolI ToolName[:[Tool Argument[:]]]

This is in BNF format

where

ToolName name of the tool (basename of a .tol file)

May not contain space.

Tool Argument (Optional) The argument passed to toolpad.

May contain data attribute name prefixed by

'$' (e.g. $Data2). In the runtime, the full

path to the data is used if a data attribute is referenced.

Example:

The attributes defined in workflow process DesignEntry:

```
================
attribute( 'Data1', 'Schematic DB::Input:::delete'),
attribute( 'DataList', 'Data2:Data1:Data3:Data4:InOut'),
attribute( 'Data2', 'Requirement Spec:Final:Control'),
attribute( 'ToolList', 'Tool1:Tool2'),
attribute( 'Tool1', 'xterm:-e vi $Data1:'),
attribute( 'Tool2', 'xv'),
attribute( 'Data3', 'A File:Modified Once:Output'),
attribute( 'Data4', 'Another Input::Input:SharedWL:checkout'),
attribute( 'Status', '0:'),
attribute( 'InOut', 'A File:Initial:Input')
================
```

The template script:

```
================
#!/bin/sh
# Assumption
# each input item is of Class Document
#
# Input
#
```

```
# Exit status
# 0 --- successful
# 1 --- error copying input from source vault to user's private work
# location
#
#
# Step 1
# DMM changed the ND_PATH, add the nd path back
ND_PATH=$RASSP_CUSTOM_PATH/ingr/nd:$ND_PATH; export ND_PATH
#
#
# Step 2
# Don't change the following line!
# Input/Control gathering section:
#
#
# Step 3
# Don't change the following line!
# Tool Launch Pad section:
#
#
# Step 4
# Don't change the following line!
# Output verification section:
#
# Don't change the following line!
# Script status section:3
#
#
```

```
# Step 5
# Don't change the following line!
# Input/Control restoring section:
#
# Don't change the following line!
# Script status section:4
#
#
# Step 6
# Don't change the following line!
# DMM notification section:
#
# Don't change the following line!
# Script status section:0
#
exit 0
```

===================

Syntax:

===================

```
Usage: autoscr [-h] -d dir -p proc [-t temp_scr] [-o save_scr]
-h: print this message.
-d dir: the dir where the workflow attached to.
-p proc: name of the process from which this application is
invoked. Put proc in quotes if it contains spaces.
-t tem_scr: template script file name.
This overrides the env RASSP_TEMPLATE_SCRIPT.
```

-o save_scr: save a copy of the script to save_scr.

================

The output script generated:

================

```
#!/bin/sh
# Assumption
# each input item is of Class Document
#
# Input
#
# Exit status
# 0 --- successful
# 1 --- error copying input from source vault to user's private work
# location
#
#
# Step 1
# DMM changed the ND_PATH, add the nd path back
ND_PATH=$RASSP_CUSTOM_PATH/ingr/nd:$ND_PATH; export ND_PATH
#
#
# Step 2
# Don't change the following line!
# Input/Control gathering section:
Data2="Requirement Spec"
```

```
Data2_PATH=`get_data reference SharedVL "$Data2"`
if [ $? -ne 0 ]; then exit 1; fi
Data1="Schematic DB"
Data1_PATH=`get_data copy Private "$Data1"`
if [ $? -ne 0 ]; then exit 1; fi
Data4="Another Input"
Data4_PATH=`get_data checkout SharedWL "$Data4"`
if [ $? -ne 0 ]; then exit 1; fi
InOut="A File"
InOut_PATH=`get_data checkout Private "$InOut"`
if [ $? -ne 0 ]; then exit 1; fi
#
#
# Step 3
# Don't change the following line!
# Tool Launch Pad section:
TOOL_ARG="xterm(-e vi $Data1_PATH) xv() "
toolpad -d "/tmp_mnt/opt/dm2.0/jjyou/toolpad" -p "DesignEntry" -t "$TOOL_ARG"
if [ $? -ne 0 ]; then exit 1; fi
#
#
# Step 4
# Don't change the following line!
# Output verification section:
put_data checkin SharedWL "A File"
if [ $? -ne 0 ]; then exit 1; fi
#
# Don't change the following line!
# Script status section:3
```

```
dmm_attr_set -d "/tmp_mnt/opt/dm2.0/jjyou/toolpad" -p "DesignEntry" -v 3
#
#
# Step 5
# Don't change the following line!
# Input/Control restoring section:
put_data delete Private "$Data1_PATH"
if [ $? -ne 0 ]; then exit 1; fi
put_data checkin SharedWL "Another Input"
if [ $? -ne 0 ]; then exit 1; fi
#
# Don't change the following line!
# Script status section:4
dmm_attr_set -d "/tmp_mnt/opt/dm2.0/jjyou/toolpad" -p "DesignEntry" -v 4
#
#
# Step 6
# Don't change the following line!
# DMM notification section:
notif_dmm -d "/tmp_mnt/opt/dm2.0/jjyou/toolpad" -p "DesignEntry"
if [ $? -ne 0 ]; then exit 1; fi
#
# Don't change the following line!
# Script status section:0
dmm_attr_set -d "/tmp_mnt/opt/dm2.0/jjyou/toolpad" -p "DesignEntry" -v 0
#
exit 0
================
```

Comments:

DataList includes 4 items: Data1, Data2, Data3, Data4, InOut

Data1 is type Input, no Retrieve specified, so the default is copy,
no destination specified, the default is Private. This
reflects in Step 2:

```
Data1="Schematic DB"
Data1_PATH=`get_data copy Private "$Data1"`
if [ $? -ne 0 ]; then exit 1; fi
```

The dispose delete overrides the default dispose delcopy, in
Step 5:

```
put_data delete Private "$Data1_PATH"
if [ $? -ne 0 ]; then exit 1; fi
```

Data2 is Control, no Retrieve specified, so the default is reference,
no destination specified, the default is SharedVL. In Step 2:

```
Data2="Requirement Spec"
Data2_PATH=`get_data reference SharedVL "$Data2"`
if [ $? -ne 0 ]; then exit 1; fi
```

Since the Retrieve is reference, the dispose is default to
nochange, therefore no Retrieve is taken in Step 5.

Data4 is another input, checkout is the Retrieve that overrides default copy, the destination SharedWL overrides the default Private. In Step 2:

```
Data4="Another Input"
Data4_PATH=`get_data checkout SharedWL "$Data4"`
if [ $? -ne 0 ]; then exit 1; fi
```

Since Data4 is checkout, it needs to be checkin. So in Step 5:

```
put_data checkin SharedWL "Another Input"
if [ $? -ne 0 ]; then exit 1; fi
```

Inout is Input with state Initial
Data3 is Output with state Modified Once
Since the output Data3 is modified from the input InOut, the Retrieve for InOut is set to checkout, therefore in Step 2:

```
InOut="A File"
InOut_PATH=`get_data checkout Private "$InOut"`
if [ $? -ne 0 ]; then exit 1; fi
```

The destination for Data3 is set to the same as InOut, which is Private. The output Data3 is checked-in in Step 4:

```
put_data checkin Private "A File"
if [ $? -ne 0 ]; then exit 1; fi
```

ToolList includes 2 items: Tool1 and Tool2

Tool1 has argument "-e vi $Data1", which refering to data item Data1. As the result it is expanded to "-e vi $Data1_PATH" when it is used in the toolpad invocation. The Data1_PATH is set as the result of get_input.

Tool2 does not have arg, so it is simply passed to toolpad.

Examples of Script flow control

1. Suppose Steps 1-4 was finished successfully in the first run, then in the second run, the script becomes

```
==================
#!/bin/sh
# Assumption
# each input item is of Class Document
#
# Input
#
# Exit status
# 0 --- successful
# 1 --- error copying input from source vault to user's private work
# location
#
#
# Step 1
# DMM changed the ND_PATH, add the nd path back
ND_PATH=$RASSP_CUSTOM_PATH/ingr/nd:$ND_PATH; export ND_PATH
#
#
# Step 2
# Don't change the following line!
# Input/Control gathering section:
Data2="Requirement Spec"
Data2_PATH=`get_data reference SharedVL "$Data2"`
if [ $? -ne 0 ]; then exit 1; fi
Data1="Schematic DB"
Data1_PATH=`get_data copy Private "$Data1"`
if [ $? -ne 0 ]; then exit 1; fi
Data4="Another Input"
Data4_PATH=`get_data checkout SharedWL "$Data4"`
if [ $? -ne 0 ]; then exit 1; fi
#
#
# Step 3
# Don't change the following line!
# Tool Launch Pad section:
TOOL_ARG="xterm(-e vi $Data1_PATH) xv() "
toolpad -d "/tmp_mnt/opt/dm2.0/jjyou/toolpad" -p "DesignEntry" -t "$TOOL_ARG"
if [ $? -ne 0 ]; then exit 1; fi
#
```

```
#
# Step 4
# Don't change the following line!
# Output verification section:
#
# Don't change the following line!
# Script status section:3
dmm_attr_set -d "/tmp_mnt/opt/dm2.0/jjyou/toolpad" -p "DesignEntry" -v 3
#
#
# Step 5
# Don't change the following line!
# Input/Control restoring section:
put_data delete Private "$Data1_PATH"
if [ $? -ne 0 ]; then exit 1; fi
put_data checkin SharedWL "Another Input"
if [ $? -ne 0 ]; then exit 1; fi
#
# Don't change the following line!
# Script status section:4
dmm_attr_set -d "/tmp_mnt/opt/dm2.0/jjyou/toolpad" -p "DesignEntry" -v 4
#
#
# Step 6
# Don't change the following line!
# DMM notification section:
notif_dmm -d "/tmp_mnt/opt/dm2.0/jjyou/toolpad" -p "DesignEntry"
if [ $? -ne 0 ]; then exit 1; fi
#
```

```
# Don't change the following line!
# Script status section:0
dmm_attr_set -d "/tmp_mnt/opt/dm2.0/jjyou/toolpad" -p "DesignEntry" -v 0
#
exit 0
=================
```

Comment: the following lines are skipped in Step 2:

```
InOut="A File"
InOut_PATH=`get_data checkout Private "$InOut"`
if [ $? -ne 0 ]; then exit 1; fi
```

because the output Data3 is modified from InOut, and had been generated successfully in the first run.

The following lines are skipped in Step 4:

```
put_data checkin SharedWL "A File"
if [ $? -ne 0 ]; then exit 1; fi
```

because the output Data3 had been generated successfully in the first run.

2. Suppose Steps 1-5 was finished successfully in the first run, then in the second run, the script becomes

```
=================
#!/bin/sh

# Assumption

# each input item is of Class Document

#

# Input

#

# Exit status

# 0 --- successful

# 1 --- error copying input from source vault to user's private work

# location

#

#

# Step 1

# DMM changed the ND_PATH, add the nd path back

ND_PATH=$RASSP_CUSTOM_PATH/ingr/nd:$ND_PATH; export ND_PATH

#

#

# Step 2

# Don't change the following line!

# Input/Control gathering section:

#

#

# Step 3

# Don't change the following line!

# Tool Launch Pad section:

#

#

# Step 4

# Don't change the following line!
# Output verification section:
#
# Don't change the following line!
# Script status section:3
dmm_attr_set -d "/tmp_mnt/opt/dm2.0/jjyou/toolpad" -p "DesignEntry" -v 3
#
#
# Step 5
# Don't change the following line!
# Input/Control restoring section:
#
# Don't change the following line!
# Script status section:4
dmm_attr_set -d "/tmp_mnt/opt/dm2.0/jjyou/toolpad" -p "DesignEntry" -v 4
#
#
# Step 6
# Don't change the following line!
# DMM notification section:
notif_dmm -d "/tmp_mnt/opt/dm2.0/jjyou/toolpad" -p "DesignEntry"
if [ $? -ne 0 ]; then exit 1; fi
#
# Don't change the following line!
# Script status section:0
dmm_attr_set -d "/tmp_mnt/opt/dm2.0/jjyou/toolpad" -p "DesignEntry" -v 0
#
exit 0
================
```

Comment: effectively only the Step 6 is kept in this script,

because all the previous steps have been finished

successfully.

9.5 Tool File Conventions

I. Format of a tool file:

```
Tool File Format 1.0.0
tool_name:
$TOOL_ARG_LIST
arg1_name:arg1_value:
arg2_name:arg2_value:
...
$END
displayed tool name:tool_path_name::icon_file:command argument:
$TOOL_ENV_LIST
env1_name:env1_value::
env2_name:env2_value::
...
$END
```

II. Example: content of tool file da.tol

```
Tool File Format 1.0.0
da:
$TOOL_ARG_LIST
MentrDir::
```

```
A_Label:default label:
$END
MGC Design Architect ($A_Label):new_da::da.bmp:$MentrDir:
$TOOL_ENV_LIST
MGC_LOCATION_MAP:$MentrDir/mgc_loc_map::
$END
```

III. Comments:

1. tool file must have extension .tol

e.g. da.tol

2. all line starts from the first column

3. fields are separated by ':', therefore no ':' is allowed in any field

4. tool_name ususlly is the base name of the tool file

e.g. da vs da.tol

5. the number of arguments in TOOL_ARG_LIST section indicates the number of arguments required when the tool is invoked. The values of the arguments will be passed in when the tool is invoked. The order of the pass-in arguments is the order of arguments in TOOL_ARG_LIST section.

6. arguments in TOOL_ARG_LIST section can be referenced in fields 'displayed tool name', and/or 'command argument', and/or 'env_value' with the format '$arg_name'. The value of the argument will be used

wherever it is referenced.

e.g. argument MentrDir is referenced as $MentrDir, as the result
$MentrDir will be replaced by the value of argument MentrDir,
whatever it will be, in the runtime

7. no space is allowed in fields arg_name, tool_path_name, icon_file, env_name

8. the field 'displayed tool name' will be used to describe the tool in
the tool launch pad. If multiple instances of a tool is needed for
different data/argument, it would be helpful to use a label in the
'displayed tool name' to distinguish one from the other.

e.g. if the da tool is invoked for 2 schematics: sch1 and sch2,
then invoke the toolpad with two instances of da tool and
pass in sch1 and sch2 as the second argument (A_Label) of
da instance1 and da instance2, respectively. As the result,
the icons in the toolpad will be labeled as 'MGC Design
Architect (sch1)' and 'MGC Design Architect (sch2)',
respectively.

9. tool_path_name is the executable path name. full path is not necessary
if the tool_path_name is in the search PATH

10. the final command line will be formed as
'tool_path_name' 'command argument'

11. the env_name are set to env_value before the tool is invoked and they
become part of the environment variables

A.5 UYS-2A Upgrade Processor Requirements

The AN/UYS-2A upgrade development Requirements are described below. The tasks describe the technical effort and delivery schedules of all items shall be completed in 20 months after contract award. The current AN/UYS-2A configuration consists of an FPAP functional element (FPAP/FE), which is composed of three modules. Any FPAP/FE shall be replaceable with an equivalent FPCAP/FE. The upgraded AN/UYS-2A upgrade deliverable assembly shall consist of a functional element comprising two (2) floating point commercial arithmetic processor (FPCAP) modules, and one (1) floating point controller (FPCTL) with application and development software.

At the beginning of the contract the Government shall supply to the Contractor operational non-real-time C code of the ETC algorithm. Within the first 12 months of the contract the Government shall supply to the Contractor an updated algorithm to be implemented in real time software by the Contractor. Government shall supply a stimulus data set along with a set of expected results from the data using the ETC algorithm. This data set shall be run on the FPCAP/FE by the Contractor as part of acceptance testing.

Architecture Development

Contractor shall develop end-user requirements that define the system level requirements of the AN/UYS-2A upgrade. These system level requirements include algorithm, external interface, control, and BIT requirements. Contractor shall convert these system level requirements into a set of processing requirements (functional and performance). Contractor shall conduct simulations and architecture trade studies in order to allocate the processor requirements to different hardware and software elements. Contractor shall document these allocated requirements in specifications for each of the hardware and software elements.

Contractor shall analyze the Navy ETC algorithm for processing thruput and memory requirements. The algorithm hierarchy is described in the ETC preliminary RDD documentation[3]. Figure 2.1-1 is a schematic of the ETC algorithm architecture showing the possible data flow paths.

Figure 2.1-1 ETC Algorithm Architecture

Contractor shall generate Data Flow Graph (DFG) in PGM from the algorithm description.

Contractor shall consider the effect of design alternatives upon the total AN/UYS-2 life cycle cost. The Architecture Design Document (ADD) shall contain a section that includes support and maintenance philosophy together with estimates of their costs.

Contractor shall conduct simulations to verify that all hardware and software requirements are met. A VHDL based virtual prototype model will be developed for the system.

Contractor shall coordinate with the ALFS system integrator as necessary to ensure a seamless integration of the UYS-2A processor upgrade into ALFS.

Contractor shall define a test philosophy for ensuring that the FPGAs, modules and integrated system are testable. It will describe what test types are appropriate such as hardware test points, embedded firmware test routines, JTAG schemes, built-in self test, and test adapters.

Contractor shall perform a feasibility analysis for thermal, reliability, testability, maintainability, repairability, upgradeability, and NRE, manufacturing, and life-cycle costs. A risk assessment will be done of the selected implementation approach for each component based on experience, complexity, schedule criticality, and design challenge.

Hardware Development

Contractor shall design and fabricate three functional sets of the FPCAP/FE enabling the AN/UYS-2A processing assembly to be upgraded in processing capacity and input/output bandwidth. The FPCAP/FE shall be able to replace or be inserted in any slot within the AN/UYS-2A assembly that is currently designated for a FPAP/FE. The FPCAP/FE specifications shall be accomplished according to the top-level design architecture and specifications as derived in section 2.1.

Contractor shall develop the detailed design information necessary for the manufacture of the FPCAP and FPCTL modules. This information will be captured in a Detailed Design Document. Contractor shall develop test plans and procedures for the FPCAP and FPCTL modules. Test vectors will be developed for the FPCAP and FPCTL modules in accordance with the test plans and procedures.

Contractor shall build or modify existing test fixtures to test/verify the requirements of the AN/UYS-2A upgrade. All FPCAP/FE modules shall be tested for functional and operational performance, verifying processing power, software control, communication bandwidth, power dissipation, and weight.

Software Development

Contractor shall implement in the AN/UYS-2A upgrade, the system interface software needed to utilize effectively the EMSP Common Operational Software (ECOS) [2] and RASSP run time software.

Contractor shall develop software supporting the seamless integration of the applications developed in the RASSP domain for the FPCAP/FE platform and the AN/UYS-2A application development environment and run-time support domains. Contractor shall modify RASSP autocoding tools to optimize their use for implementing the FPCAP/FE features. Contractor shall develop FPCAP/FE support interfaces for the of the FPCAP/FE AN/UYS-2A PIDs on the FPCAP/FE modules.

Contractor shall code and implement in the FPCAP/FE the software control and interface functions needed to operate the functional element under the ECOS enhanced with the RASSP methodology.

Contractor shall code and implement the system test and diagnostic software pertaining to the FPCAP/FE. Contractor shall implement the system and module test and diagnostic software. It shall include comprehensive built-in-tests (BITs) addressing each major hardware item of the FPCAP/FE.

Contractor shall develop real time application software of the ETC algorithm.

AN/UYS-2A Upgrade Module and System Integration and Test

Contractor shall test the FPCTL and FPCAP modules according to documented test plans and procedures and document test results.

Contractor shall test the FPCAP/FE within the AN/UYS-2A enclosure verifying the correct interaction with the other modules and subassemblies within the system.

Environmental, Mechanical, QA, & Reliability

Contractor shall perform testing to verify the environmental constraints of the AN/UYS-2A upgrade are met as established in section 2.1.

Contractor shall perform mechanical, thermal, and environmental analysis.

Contractor shall keep logs of operational time and all failures of FPCAP after completion of engineering evaluation to make reliability growth predictions.

AN/UYS-2A Upgrade Acceptance Tests

Contractor shall perform system level end-to-end formal tests verifying functional and performance compliance to system to be conducted at a U.S. Navy designated site and witnessed by U.S. Navy representatives.

The Government will be performing temperature and vibration tests on the UYS-2 system with upgraded processor, Contractor will supply nominal engineering support to this testing if required.

A.6 Example UYS-2 Schedule

The attached MS Project Plan is included for reference - Note - it may be out of date relative to the project plan in use on the program. This represents the project plan for the workflow represented in figure 2.2-2, together with other aspects of the program.