Rapid Prototyping of Application-Specific Signal Processors (RASSP)

Build 2 System Description

Version 1.0

Lockheed Martin Corporation
Advanced Technology Laboratories
1 Federal Street A&E 2W
Camden, NJ 08102

TABLE OF CONTENTS

ACRONYMS AND ABBREVIATIONS

ARPA	Advanced Research Projects Agency
ASEM	Application Specific Electronic Module
ASIC	Application Specific Integrated Circuit
ATP	
CAD	Computer-Aided Design
CAM	Computer-Aided Manufacturing
CE	Concurrent Engineering
CM	Configuration Management
DM	Document Management
DMM	Design Methodology Manager
DSP	Digital Signal Processor
EDIF	Electronic Design Interchange Format
EF	Enterprise Framework
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IPPD	Integrated Product/Process Development
MCM	Multi-Chip Module
MRE	Manufacturing Resource Editor
NC	Numeric Control
PCA	Printed Circuit Assembly
PDES	Product Data Exchange using STEP
PGP	Pretty Good Privacy
RASSP	Rapid Prototyping of Application-Specific Signal Processors
RDOCH	Reuse Design Object Class Hierarchy
RRDMRASSP	Reuse Data Manager
SCRA	South Carolina Research Authority
SQ	Saved Query
SSL	Secure Sockets Layer
STEP	Standard for the Exchange of Product Data
TCP/IP	Transmission Control Protocol/Internet Protocol
TO	Transfer Ownership
TRP	Technology Reinvestment Program
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
UDP	User Datagram Protocol
VL	Vault Location
WL	Work Location
WS	Workspace
WWW	World Wide Web

LIST OF FIGURES & TABLES

1.0 Introduction

The purpose of this document is to describe the Build 2 version of the RASSP Enterprise Framework. The design tool encapsulation and working environments guides are included as appendices to this document.

2.0 Integrated Process and Data Management System

The RASSP Enterprise Framework contains an integrated process and data management system which manages both the design process and the product data, produced during the design of a Digital Signal Processor (DSP) system. The design process is managed by the use of workflows which guide the designer through the design process. Intergraph's Design Methodology Manager (DMM) tool is used to develop and execute the workflows. The product data is managed according to the RASSP Configuration Management (CM) model which provides for the management of the various versions of the design objects that are created and manipulated during the process of the design of a product. Intergraph's Document Manager (DM2) tool is used to implement the RASSP CM model and consequently manage the product data. DM2 is based on technology from Metaphase which offers object modeling, a relational database, rules-based processing for precise business modeling, a scaleable architecture, and true routing capabilities.

2.1 Workflows

The following workflows have been developed for the Build 2 version of the RASSP Enterprise Framework:

- Architecture Definition
    - Functional Design
    - Architecture Selection
    - Architecture Verification

- Detailed Design
    - ASIC Design
    - Backplane Design
    - FPGA Design
    - Module Design

2.2 Configuration Management Model

Configuration Management in the RASSP Enterprise System is the management of the versioning of design objects. It includes creating, approving and releasing a new version of a design object, organizing the versions of a design object, and assembling compatible configurations of versions of design objects to form a release of a product. The RASSP CM model is implemented using Intergraph's DM2 tool.

2.2.1 Shared and Private Workspaces

Three types of workspaces exist in the RASSP CM model: *private*, *shared*, and *global*. These workspaces are organized in a hierarchical manner as shown in figure 1. The global workspace is at the root of the hierarchy. The shared workspace is at the intermediate level, and the private workspace is at the leaf level. Branches in the hierarchy represent a parent-child relationship between workspaces.

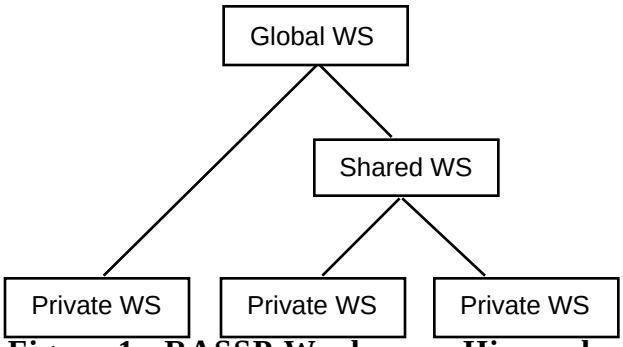

Figure 1. RASSP Workspace Hierarchy

The above workspace hierarchy is implemented in DM2 using the features of *vaults* and *users*. Parent-child relationships between workspaces are enforced by defining *groups* which contain related users, and limiting the access of these groups through the use of *rules*. Figure 2 contains the DM2 implementation of the RASSP workspace hierarchy.

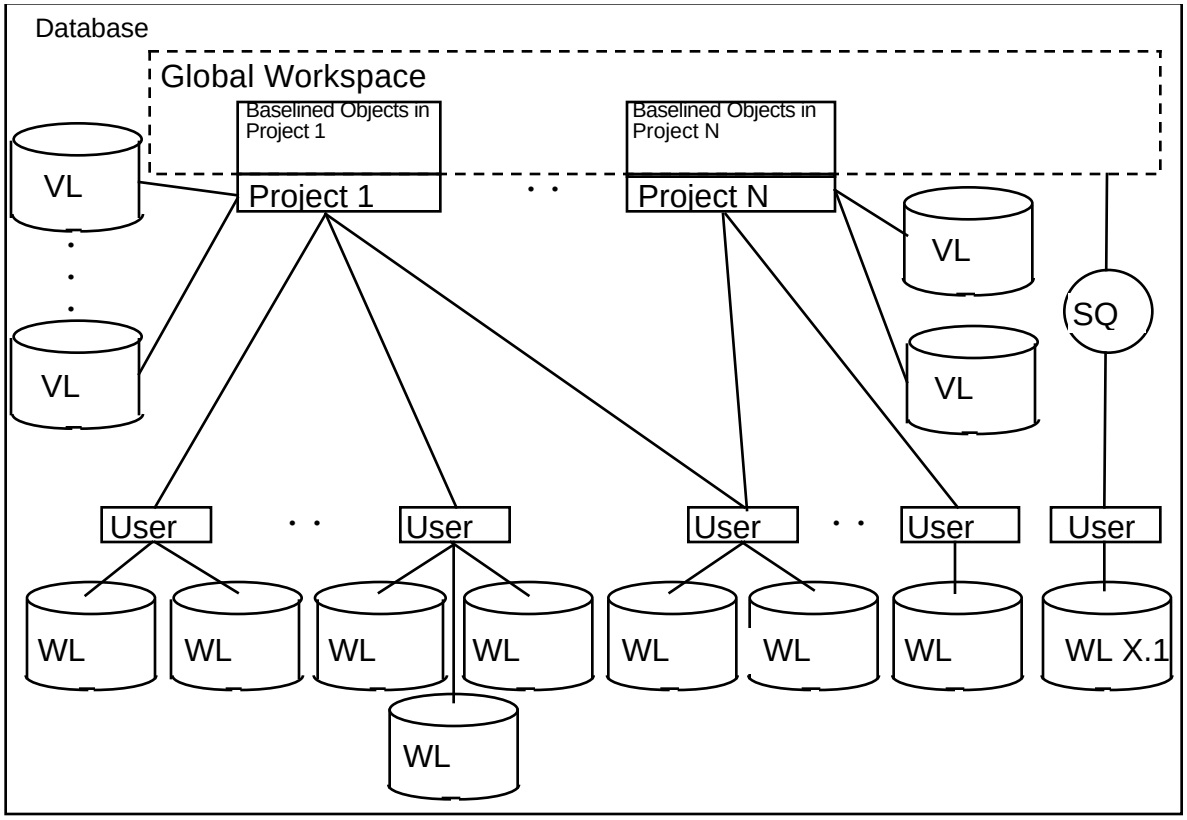

Figure 2. DM2 Implementation of RASSP Workspace Hierarchy *

***** 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 DM2 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.

*The version numbers used in the above figure do not reflect the DM2 naming convention, but are used to provide clarity to the figure***.**

In DM2, each user has a private workspace. That is, there is a one-to-one mapping between a user and a private workspace. Rules are used to enforce the privacy of individual workspaces. Shared workspaces are implemented through the use of vaults. A vault is a logical collection of shared objects. Rules are used to control access to a vault. User-to-vault relationships can be established to allow visibility of ancestor workspaces. The global workspace consists of selected data from all shared workspaces (vaults) obtained through the DM2 *query* capability.

2.2.2 Data Object Versioning

The RASSP CM model proposes 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 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 where it can be shared with other users. Working versions reside in a shared workspace. At this state, the configuration cannot be updated, but may be deleted. A configuration is considered to be in the released state when it's promoted to the global workspace. Released configurations cannot be updated or deleted.

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

The RASSP versioning scheme is implemented using the features of DM2, as shown in figure 3. DM2 is capable of managing individual objects as well as collections of objects. Hence, a configuration may be a group of one or many objects. For purposes of documenting an implementation of the RASSP CM model in DM2, 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.

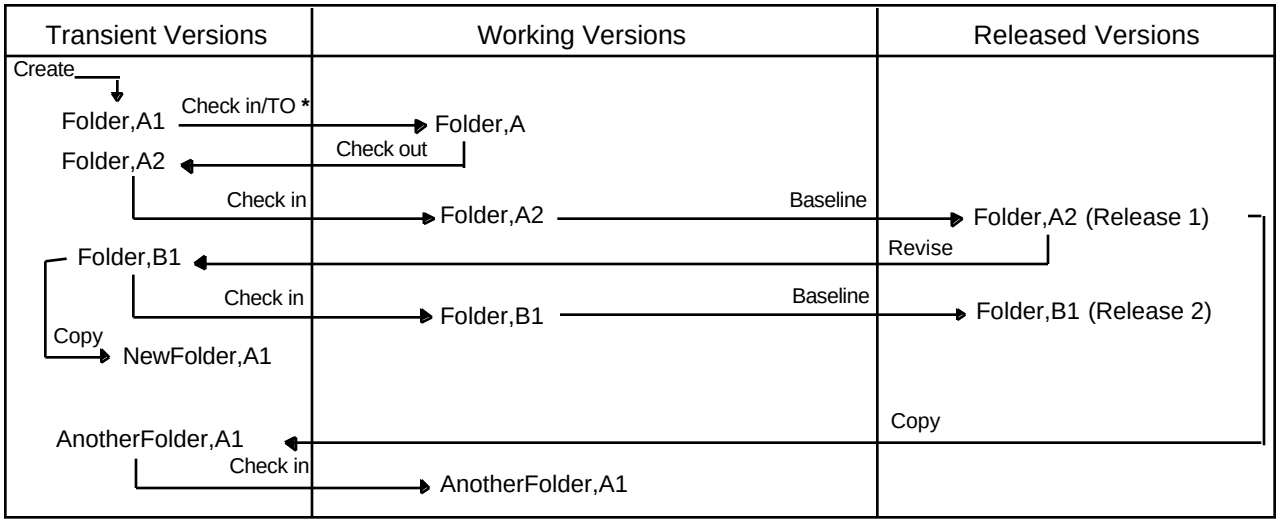

Figure 3. DM2 Implementation of Data Object Versioning

***** Transfer Ownership (TO) may be used to promote a folder to a working version the first time.
****** DM2 uses a letter-number combination to reference a folder's Revision and Sequence respectively.

A folder will be considered transient it it's 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 DM2 *create*menu, in a user's 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 promotes 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.

2.2.3 CM Functions

2.2.3.1 Workspace Functions

A *global workspace* in DM2 is mapped to all baselined objects in all vaults within a database. Items are visible in a global workspace by use of the DM2 Saved Query Object Class.
A *shared workspace* in DM2 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 DM2, 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.

2.2.3.1.1 Creating a workspace

In DM2, 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 DM2 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.

2.2.3.1.2 Accessing an arbitrary workspace

In DM2, 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 DM2 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).

2.2.3.1.3 Accessing child workspaces

In DM2, the workspace hierarchy is implemented as relationships defined between an individual user's private workspace and that user's 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 DM2 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.

2.2.3.1.4 Accessing the parent workspace

The DM2 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 DM2 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 DM2 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).

2.2.3.1.5 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 DM2. 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.

2.2.3.2 Version Management Functions

The RASSP concept of a configuration will be implemented in DM2 as a folder.

2.2.3.2.1 Creating a configuration

In DM2, 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 user's private workspace). The following steps/concepts will guide the DM2 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
- user's will be restricted to folder creation within their private workspace areas only.

2.2.3.2.2 Inserting data objects into a configuration

Once a folder is created in DM2, 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 DM2 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.

2.2.3.2.3 Check out

DM2 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 DM2 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.

2.2.3.2.4 Check in

DM2 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 it's 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 "promoted" to the global workspace by using the DM2 *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 DM2 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 folder's 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.

2.2.3.2.5 Accessing child versions

The DM2 expand relationship feature may be used to show a folder's relationships to other objects. DM2 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, DM2 can also provide a "tree" like view of these relationships. Child versions of a folder could be obtained by expanding the "is superseded by" relationship. The following steps/concepts will guide the DM2 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.

2.2.3.2.6 Accessing parent versions

The DM2 expand relationship feature may be used to show a folder's relationships to other objects. DM2 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, DM2 can also provide a "tree" like view of these relationships. Parent versions of a folder could be obtained by expanding the "supersedes" relationship. The following steps/concepts will guide the DM2 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.

2.2.3.2.7 Naming versions

DM2 will not allow the name attribute of a folder to be directly changed. Renaming a folder can be indirectly accomplished by using the DM2 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 DM2 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.

2.2.3.2.8 Retrieving a named version

The DM2 *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 DM2 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.

2.3 Authorization Model

An authorization is a triplet $\{o_i, r_j, t_k\}$ where o_i is an authorization object in an authorization object hierarchy, r_j is an authorization role in an autorization role hierarchy, and t_k is an authorization type in an authorization type hierarchy. An *authorization object* is a data object on which an authorization may be specified. An *authorization role* is a collection of users that have the same set of authorizations on the same set of objects. An *authorization type* is a type of operation that may be performed on a data object. The hierarchy for each member of the triplet is organized as a directed acyclic graph.

The directed links between the nodes in the hierarchy represent an *implication relationship* between the nodes. The hierarchy for objects and types implies inheritance from parent node to child node. The hierarchy for roles describes a structure where the parent node authorizations are equal to, or greater then the child node.

An authorization may be *positive*, granting an authorization, or *negative*, revoking an authorization. An explicit or an implicit positive authorization $\{o_i, r_j, t_k\}$ has to exist for an operation of the type t_k to be performed by a user belonging to role r_j on a data object belonging to the authorization object o_i. A positive (or negative) authorization specified on a node n_i in an authorization hierarchy may be overridden by a negative (or positive) authorization on a node n_j that follows n_i in the authorization hierarchy.

The authorization object hierarchy and the authorization role hierarchy for a project may be customized by a RASSP user/systems administrator. The authorization type hierarchies, however, are not customizable by a RASSP user/system administrator. Defining a new authorization type will typically involve adding a new functionality to the system.

2.3.1 Authorization Model in DM2

The RASSP Authorization Model can be implemented using DM2. An authorization in DM2 performs the same function as the triplet described above, $\{o_i, r_j, t_k\}$ where o_i is an authorization object in an authorization object hierarchy, r_j is an authorization role in an authorization role hierarchy, and t_k is an authorization type in an authorization type hierarchy. DM2 extends the definition of the triplet by adding a *condition* which defines the circumstances that allow the authorization role to perform the authorization type on the authorization object. An authorization in DM2.0 can then be described as a quadruplet $\{o_i, r_j, t_k, c_n\}$, where c_n is the *authorization condition*.

In DM2 the quadruplet is called *a message access rule*. An authorization object in a message access rule is an object *class* on which an authorization may be specified. An authorization role in a message access rule is a defined *group* or *role* for which the authorization is valid. An authorization type in a message access rule is a *message group* that defines the operations that may be performed on the object class.

A message access rule may be described in the following way: a rule grants permission for the stated group, or role, to send a message from the stated message group to the stated object class under the defined condition.

For example:

Condition: IS_OWNER
Class: WorkItem
MessageGrp: UpdateGrp

Participant: Engineering Manager

This example states that the Engineering Manager role can send any message in the UpdateGrp message group to any class of items at the WorkItem node level, or below it in the object hierarchy, under the defined condition that the user in the Engineering Manager role owns that WorkItem.

In DM2 rules are only granting, or positive, in nature. An explicit rule must exist for an action to be performed on an object class. If a rule exists, then the actions that the rule grants cannot be overridden or limited by another rule. In DM2 if two or more rules conflict or overlap, then the more permissive rule is always followed.

Inheritance in DM2 is found only in the object class hierarchy. A message group defines only the messages that are availabe through that rule. Likewise, all authorizations granted to a participant must be explicitly defined for that participant. In other words, authorizations for one participant (user, role, or group) who has authority over other participants, are not necessarily a superset of authorizations defined for those under his authority. Similarily, those participants under another authority do not necessarily have a subset of authorizations defined for them. (Asterisks '*' may be used as wildcards in defining object class, participant, or message group.)

In the RASSP Enterprise Framework (EF) implementation of the DM2 authorization, the DM2admin user will be the only user who may create or update rules. The DM2admin will also be responsible for creating new groups or roles, new EF projects, and new object classes on the system.

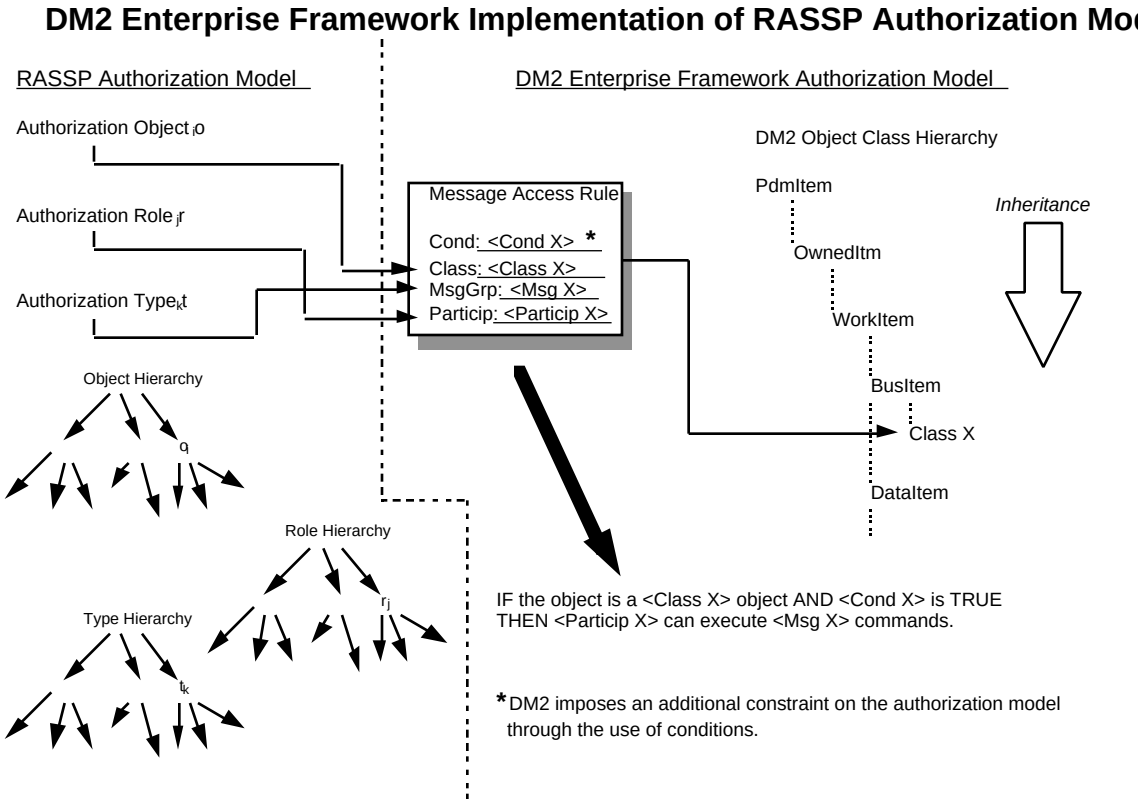

Figure 4. DM2 Implementation of RASSP Authorization Model

2.3.2 Mechanisms to Manipulate the Authorization Object Hierarchy

2.3.2.1 Creating an Authorization Object

In DM2, an object class can be created and inserted into the Object Class Hierarchy at any level except at the hierarchy root node. Adding a new object class involves many steps that are complex and will alter the customized environment where DM2 is installed. Updates made to the customized environment of DM2 should be handled by one person, namely the DM2 System Adminstrator.

2.3.2.2 Deleting an Authorization Object

In DM2 an object class can be deleted from the Object Class Hierarchy at any level except at the hierarchy root node. Deleting an object class involves many steps that are complex and will alter the customized environment where DM2 is installed. Updates made to the customized environment of DM2 should be handled by one person, namely the DM2 System Adminstrator.

2.3.2.3 Adding a Child to an Authorization Object

Adding a child to an authorization object is similar to creating a new authorization object. Since an object in DM2 may appear in only one node in the authorization object hierarchy it may be moved, but not copied to another node.

2.3.2.4 Associating Data Files with Authorization Objects

In DM2, an object class may be instantiated any number of times to create many specific objects of the same type. Once an object is created and data associated with it, that object may be handed over to configuration management.

2.3.2.5 Retrieving an Authorization Object

In DM2, a query can be made for a specific object class which will return a list of objects of that class, plus any other class below it in the object hierarchy which is currently defined in the scope of the database.

2.3.2.6 Retrieving the Children of an Authorization Object

In DM2, a query can be made for a specific object class which will return a list of objects of that class, plus any other class below it in the object hierarchy which is currently defined in the scope of the database.

2.3.3 Mechanisms to Manipulate the Authorization Role Hierarchy

2.3.3.1 Creating an Authorization Role

In DM2, authorization roles, whether they are defined roles or groups, do not have explicitly defined parent-child relationships with other authorization roles. Such a relationship is expressed through the rules that are created for each specific authorization role. One role may have authority over another, which could be interpreted as a parent-child relationship. Creating an authorization role should be the responsibilty of a specific user, thus controlling the number of roles that are created.

2.3.3.2 Deleting an Authorization Role

In DM2, deleting a role has no affect on any of its implied children. All users must be removed from the role before it is deleted and the associated rules removed also. Deleting an authorization role should be the responsibilty of a specific user, thus controlling which roles are deleted and when.

2.3.3.3 Adding a Child to an Authorization Role

In DM2, a parent-child relationship between authorization roles is implied through the rules defined for the specific roles. The rules for each role define the actions available to that role. Some roles may be granted more authority than others, essentially giving one role authority over an other. This implies a parent-child relationship.

2.3.3.4 Associating Users with Authorization Roles

DM2 provides an easy "drag-and-drop" GUI interface to associate users with authorization roles. By simply choosing a user and dragging that user into the desired role or group creates an association between that user and that role. All the rules defined for the role are defined for the users in that role.

2.3.3.5 Retrieving an Authorization Role

In DM2, a query can be made for an authorization role. This query will return the authorization being queried if it exists in the database. Since there are no explicitly defined relationships between roles, the need for a root node, which defines the tree to search, need not be given to the query.

2.3.3.6 Retrieving the Children of an Authorization Role

In DM2, a query cannot be defined to request the children of a specific role. A query for a specific role may be made, or a query for all the roles in the database may be made. But given an authorization role, there is no defining relationship between it and any implied child node.

2.3.4 Mechanisms to Manipulate the Authorization Type Hierarchy

2.3.4.1 Retrieving an Authorization Type

In DM2, a query can be defined for an authorization type. This query will return the authorization type if it exists in the database. Authorization types in DM2 can be accessed as one specifc message, a message grouping, or all the messages that exist.

2.3.4.2 Retrieving the Children of a Node in the Authorization Type Hierarchy

In DM2, a query cannot be defined to request the children of a specific authorization type. A query for a specific type may be made, or a query for all the authorization types in the database may be made. But given an authorization type, there is no defining relationship between it and any child type.

2.3.5 Mechanisms to Grant Authorizations

Since DM2 is granting by design, new authorizations cannot be created to revoke or limit existing authorizations.

2.3.5.1 Granting Authorizations

In DM2 authorizations can be defined using Message Access Rules. Given an authorization object <Class W>, an authorization role <Participant X>, an authorization type <Message Group Y>, and a condition under which the authorization is granted <Condition Z>, a Message Access Rule can be written to grant authorization as described below--
IF the object is a <Class W> object, AND <Condition Z> is TRUE THEN
<Participant X> can execute <Message Group Y> commands.

2.3.5.2 Revoking Authorizations

In DM2, the only way to revoke an authorization is to remove that authorization, or message access rule, from the database. This will remove the granting authorization.

3.0 Reuse Data Management

In today's design environments, the ability of a design engineer to maximize reuse is impaired by the fact that there is no efficient way of searching for reusable design objects across multiple sources, and that many sources of reusable data are uncoupled from the design environment. Mechanisms and processes for organizing reusable design information created within a design organization and for effectively sharing design data within the organization as well as with other cooperating organizations are also lacking. Given that we believe design reuse to be key to achieving the 4x improvement goals of the RASSP program, Lockheed Martin Advanced Technology Laboratories has developed a library management model [LM-ATL, 1995] for integrating the various sources of reusable design objects to provide a single source for searching and enable enterprise-wide sharing of reuse data. Our approach consists of (1) developing a Reuse Design Object Class Hierarchy (RDOCH) that classifies the various types of design objects in the RASSP domain and models the descriptive data associated with them, and (2) developing a commercial library management system to implement the RDOCH, providing mechanisms for searching across multiple libraries in a distributed, virtual corporation environment.

3.1 Architecture

Figure 5 shows the data flow architecture of the RASSP enterprise system. A design engineer interacts with the workflow manager to perform activities specified by a particular workflow. The workflow manager in turn invokes the appropriate CAD tool(s) for the activity, and interacts with the product data manager on behalf of the user to manage the appropriate design data. Once a particular tool has been launched, the designer interacts with that tool in its native environment to perform a design and/or analysis task. Depending on the function being performed, the designer may also invoke the RASSP Reuse Data Manager (RRDM) either through the CAD tool, through the workflow manager, or directly. Figure 5 highlights the data flow for searching for, importing, creating, and maintaining reusable design objects in the enterprise environment.

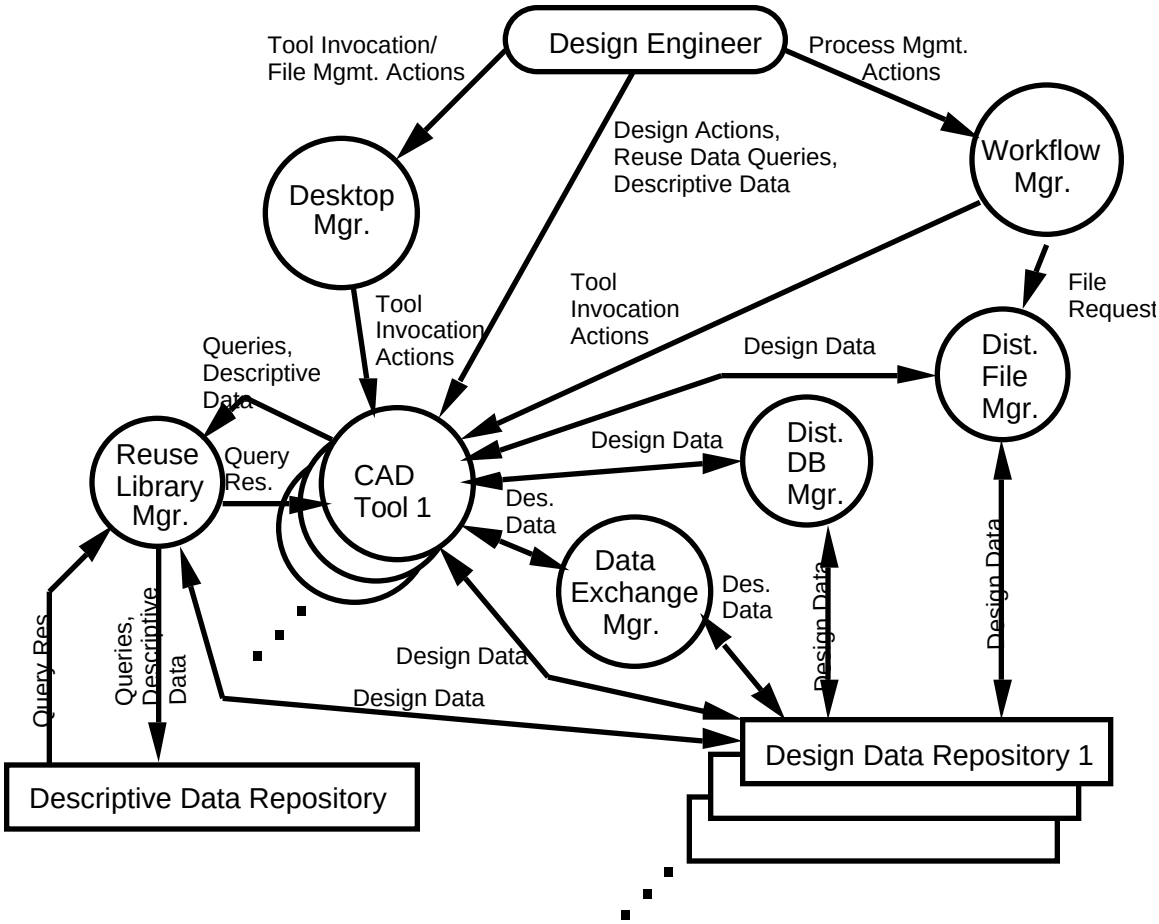

Figure 5. Rassp Enterprise System Data Flow Architecture

3.2 Library Data Management

Library management in the RASSP system involves cataloging, releasing, and searching for reusable design objects. The RRDM stores metadata describing all reusable design objects available in the RASSP environment. A designer locates reusable design objects by querying on the metadata, and may view a particular design object using a standard viewer or a viewer specific to the tool that created it, importing it into the design environment if it meets their requirements. Reusable design objects are stored in native design tool formats, or in standard interchange formats where possible.
Sources for reusable design objects in the RASSP environment include the following:

- native CAD tool libraries
- standalone tool-independent libraries
- vendor product information
- specifications and standards
- design objects created within a design organization

Physical design objects may be stored within the tool environment, in the RRDM design data repository, or in a file system within the virtual enterprise network, while the metadata describing the reusable design data are stored within the RRDM descriptive data repository. Metadata are modeled using the classes of the Reuse Design Object Class Hierarchy

(section 3.3). Figure 6 shows the default workflow to be followed to add reusable design objects to the RRDM (IDEF3 standard notation).

Key requirements for library data management identified to date include:

- provide a single source for searching for reusable design data in a fully distributed, cooperative, heterogeneous environment
- provide the capabilities to manage descriptive data about the reusable design objects, manage the objects themselves, and query for these objects in an object-oriented classification hierarchy
- enable searches for multiple views of the same design information (*e.g.*, a PGM Graph and VHDL model for the same architecture)
- enable searches based on complex relationships between design objects (*e.g.*, between the supplier, hardware module, architecture, simulation models, and schematic for the same processor board)
- provide a level of performance when querying over millions of design objects, each of which may have 100+ attributes defined for them, acceptable to a designer working from the desktop anywhere in the virtual enterprise
- provide a standards-based mechanism for tool interoperability between the library management system and other enterprise and design tools in the RASSP environment, allowing:
    - maintenance of descriptive data only in the library itself, with references to the physical design objects within the native tool environment (or in the network file system)
    - maintenance of both the descriptive data and physical design objects in the library, with the capability to view the design objects in native form (*e.g.*, CAD-tool specific viewers, word processors, etc.)
    - automated design data exchange and metadata synchronization between the library and enterprise/design tools where feasible

The Library Management Model for the RASSP System [LM-ATL, 1995], documents the reuse data management, access, and integration requirements for the program, reflecting the latest refinements in our approach.

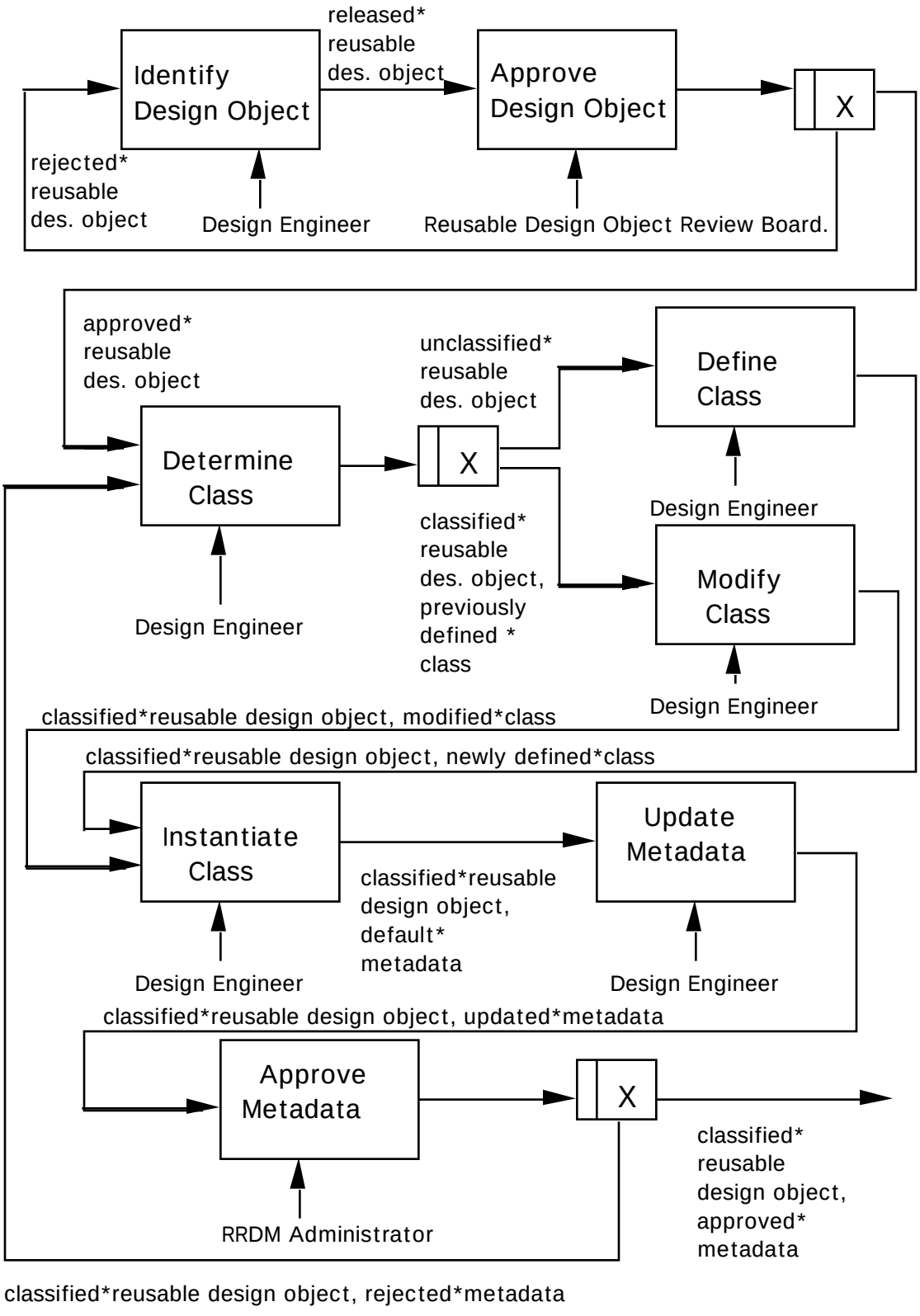

Figure 6. Workflow for Reusable Design Object Definition

Implementation of the RASSP Reuse Data Manager for Build 1 is based on the Aspect *Explore-CIS*™ tool. RASSP-supported development over the past two years includes object-oriented enhancements to the original Aspect CIS (Component Information System) product to create class browser, metadata viewing, metadata editing, and data model modification capabilities. *Explore-CIS*™ was formally announced as a commercially-available product by Aspect in May, 1995, is currently in production at several Aspect customer sites, and was demonstrated at the July '95 RASSP Annual Conference. *Explore-CIS*™ version 2.5.2 will be used on build 2 as part of the Benchmark III enterprise environment, including the baseline reference library described in section 3.3 below. Additional work is ongoing to complete integration with Mentor Graphics Library Management System (LMS) based on the RASSP-supported prototype development effort, to specify integration with the Intergraph DM2 System, and to develop read/write API and interprocess communications capabilities. The RRDM/LMS integration effort is expected to be complete for use on Benchmark III. RRDM/DM2 integration is planned for Build 2, along with additional *Explore-CIS*™ product enhancements.

3.3 Reuse Design Object Class Hierarchy

As stated in the introduction, independently of the development of the library management tool, development of a Reuse Design Object Class Hierarchy (RDOCH) as a basis for organizing the reusable design data is an essential part of the Lockheed Martin Advanced Technology Laboratories program. This effort involves identifying and using existing standards for data organization where they exist (*e.g*., IEC 1360-1 for electrical component information [IEC, 1994]), augmenting these standards as needed (as Aspect Development, Inc. has done with respect to the IEC 1360-1 standard), and creating new classification schemes where no standards exist. The overarching goal is to develop a classification scheme that characterizes all classes of reusable design data for the RASSP domain, that can be implemented in the library management system to provide a single source for searching for reuse information, and that is intuitive from a user perspective. This scheme must

- be general enough so that is can be adapted to fit most corporate environments
- provide complete, consistent, and correct classification of design data, normalized across tools and data suppliers
- be rigorously defined and reviewed by a large enough audience so that it can become the basis for an industry standard

The model must be generic, or it will not be accepted by the corporate community that we hope will benefit from its development. Consistent, standardized classification of data is a necessity, or search results may be unpredictable. Additionally, the descriptive data repository developed under RASSP must be populated sufficiently to demonstrate its utility. Ideally, the resultant library demonstration should also show how the classification hierarchy can be adapted for use in a variety of corporate environments that may use a different mix of development tools and COTS source data, or that may produce a wider variety of products than the RASSP domain addresses.

The methodology we have adopted for development of the RDOCH includes rigorous definition of preliminary and final classification trees and complete data dictionaries for each class, with review by the LM-ATL RASSP team, appropriate RASSP team members and external organizations (*e.g*., beta sites, ARL) at various phases of the development process. This methodology is described in detail in the RASSP Reuse Data Manager and

Reuse Strategy Requirements Specification [Aspect, 1995a]. The highest level of the current version of the RDOCH is shown in Figure 7.

An initial version of the RDOCH was implemented in the RASSP Reuse Data Manager and demonstrated at the July '95 Annual Conference. This version includes implementation of (1) Electrical Component classes based on the Aspect VIP 1.2 Reference Database, (2) Simulation Model data based on the RASSP taxonomy for VHDL model classification [LM-ATL *et al.*, 1995], (3) Architectural Design classes based on the LM-ATL Model Year Architecture Report [LM-ATL, 1994], and (4) Algorithm Design classes based on the Q003 Specification [AT&T, 1993], RASSP Domain Primitive Library Specification [MCCI, 1995], and appropriate tool vendor documentation. Additional work is ongoing in the areas of module and chassis/backplane design, algorithm design, software design, and supplier management for Build 2. Build 2 will also use the Aspect VIP 96.1 Reference Database. The current implementation of the RDOCH will be used in support of Benchmark III.

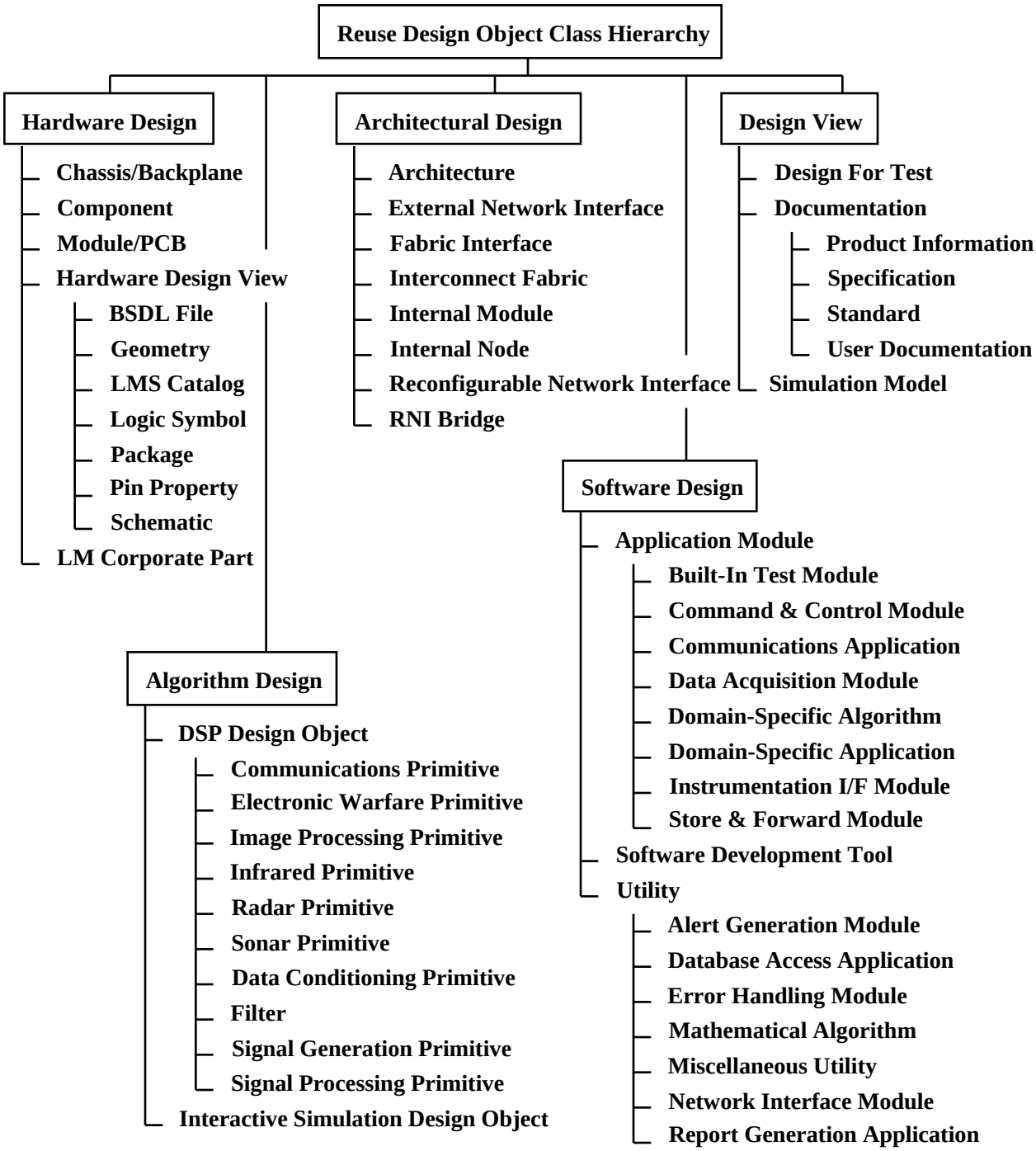

Figure 7. Reuse Design Object Class Hierarchy (Preliminary)

4.0 Manufacturing Interface

The overall mission of the RASSP Manufacturing Interface is to enable first-pass manufacturing success of application-specific signal processors. To achieve this goal, concurrent engineering techniques must be used between design and manufacturing to ensure that manufacturability is built into designs from the beginning. This level of communication and cooperation can be achieved most effectively in the context of a "virtual

enterprise". The information sharing infrastructure that forms the backbone of a virtual enterprise can only be achieved through the development, acceptance and adherence to information sharing standards such as VHDL, EDIF, and STEP.

Information sharing standards are vitally important to the successful creation of a RASSP capability (rapid prototyping coupled with concurrent engineering). These standards play three roles within a RASSP system. First, they provide the infrastructure needed to integrate different CAD/CAM tools into an automated concurrent engineering environment. Second, these same standards enable the implementation of the model year concept that is needed to manage the disparity between electronic and weapon system life-cycles. Finally, these standards enable the effective and efficient transformation of prototype information into information for production by capturing not only design information but design intent as well. This third role is critically important because rapid prototyping is only the first step in the process of rapidly producing application-specific signal processors. The implementation of a standards based Enterprise Framework within the RASSP project is crucial to the success of the overall RASSP system.

The Manufacturing Interface being developed by the SCRA Team is a critical component of the Enterprise Framework. To achieve the goal of first-pass manufacturing success, the Manufacturing Interface provides seamless integration of design and manufacturing as well as supporting Integrated Product/Process Development (IPPD). By providing an IPPD capability, the Manufacturing Interface allows design prototypes to be produced more quickly. By using a standards based interface, the RASSP Manufacturing Interface supports virtual partnering between design and manufacturing organizations. The RASSP Manufacturing Interface effort is making effective use of existing projects such as the industry funded PDES, Inc. Electrical project, the ATP-funded *P*re*A*mp program, the ARPA-funded ASEM MCM efforts, the TRP-funded CommerceNet program, and others. By leveraging existing work wherever possible, RASSP is developing a highly flexible and cost-effective solution to the manufacturing interface problem.

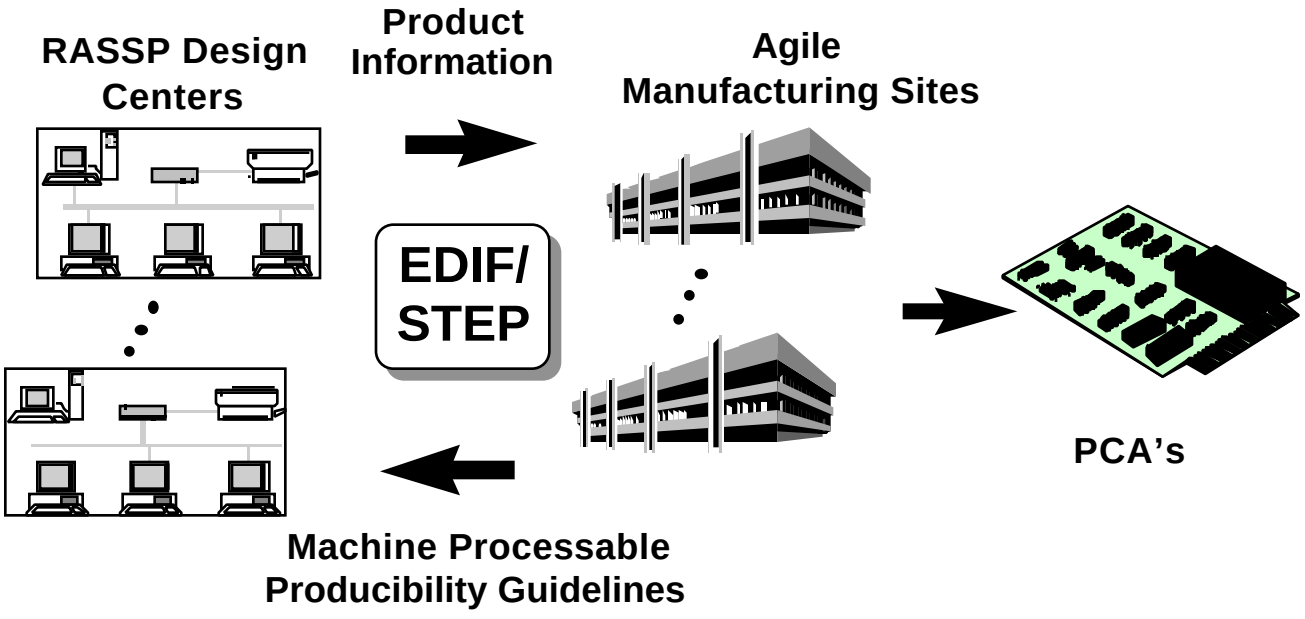

Figure 8. Manufacturing Interface Concept

At the heart of the Manufacturing Interface is a novel concurrent engineering capability. The principle focus of this capability is to enable an effective DFx capability by creating an IPPD environment. This Concurrent Engineering (CE) environment is distinguished from other CE environments in two respects. First, it utilizes a standards-based methodology to create the information sharing infrastructure necessary for IPPD. Second, it provides a unique, knowledge-centered approach to concurrent engineering. This is accomplished by integrating an inference engine into the standards-based information sharing environment. The result is an automated concurrent engineering capability. This capability allows engineers from different disciplines to capture their experience in an executable form. This executable knowledge may then be used to detect potential producibility, testability, and other "ility" issues early in the product development process.

The SCRA Team has adopted a rapid prototyping approach to develop the Manufacturing Interface capability. This development approach will result in incrementally increasing the capabilities of the RASSP Manufacturing Interface on a yearly basis. To demonstrate these capabilities, the Manufacturing Interface has been put into service at Lockheed Martin's Printed Circuit Assembly (PCA) facility in Ocala, Florida,

The Manufacturing Interface is composed of several distinct tools. The Manufacturing Resource Editor (MRE) is used to capture the capabilities of a manufacturing facility in standard form. The Mentor-to-STEP Data Converter tool is used to convert Mentor design files into standard STEP files. The STEP product data is used by a Producibility Advisor in conjunction with manufacturing capability information captured by the MRE to determine any issues against the design based on the manufacturing production line chosen. The product data and manufacturing capabilities are also used by the Process Planner to create a manufacturing process plan. Manufacturing and design issues are resolved via the Web-based Access Mechanism and collaboration tools. A secure Internet connection will be used to transfer data between design sites and the Ocala manufacturing site.

In Build 2, support for EDIF 4 0 0 is being added to the standards-based interface. Also, Mitron's CIMBridge manufacturing support system is being integrated into the Manufacturing Interface, providing more robust DFx analysis capabilities and commercially supported Numeric Control (NC) program generators. The architecture of the Manufacturing Interface is shown in Figure 9.

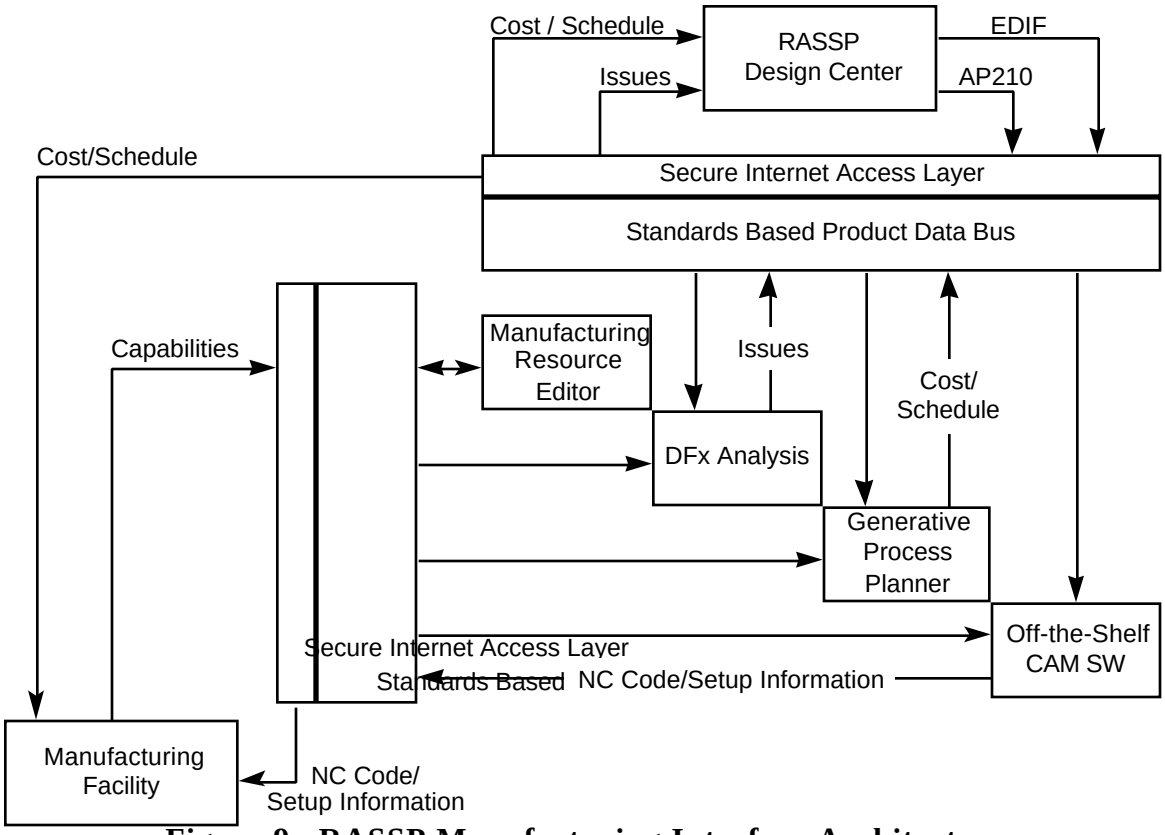

Figure 9. RASSP Manufacturing Interface Architecture

Ultimately, the capabilities embodied in the Manufacturing Interface will enable concurrent engineering in the context of the electronic commerce paradigm. Figure 10 illustrates how this will be accomplished. The initial design customer contact with the manufacturer takes place via a public World Wide Web (WWW) connection. If further contact is desired the customer registers with the manufacturer and is given access to a secure WWW server. The customer may then transfer design data via secure Internet to the manufacturing site and use a web browser to run the design data against the manufacturer's capabilities. Manufacturing issues and relative pricing are returned to the customer via the secure WWW connection. If additional manufacturing knowledge is required, the customer may request assistance from a manufacturing engineer. Additional iterations of this scenario may be required before a satisfactory design is obtained. When the design is complete, the customer contacts the manufacturing engineer for detailed cost and scheduling information and data verification.

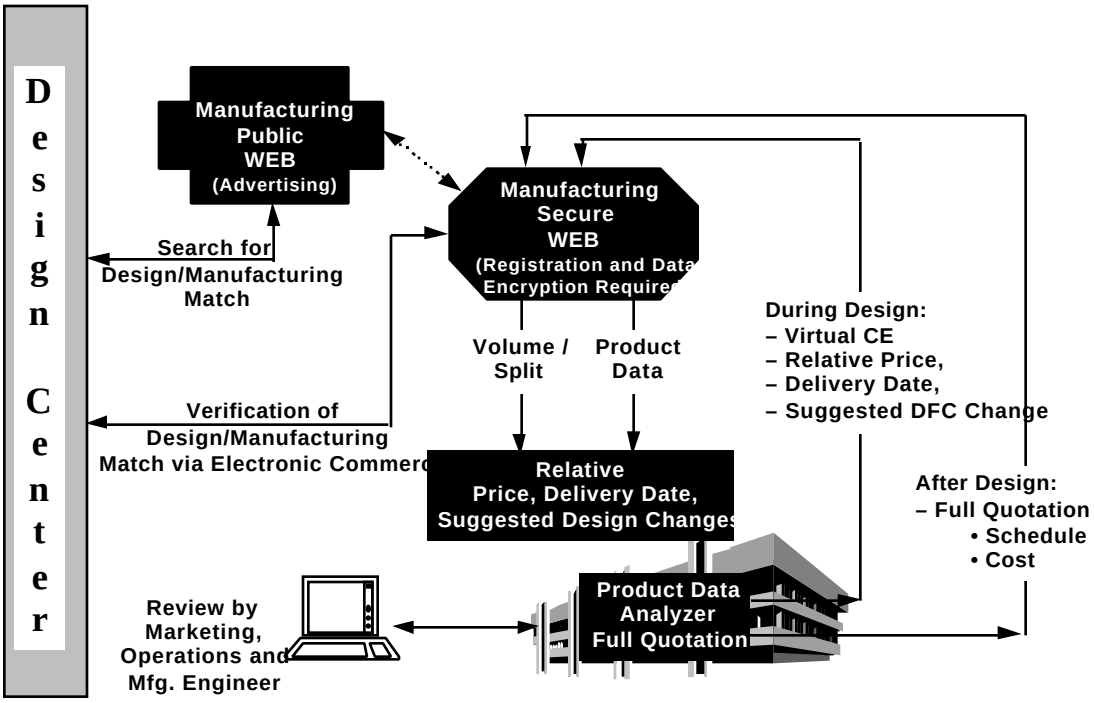

Figure 10. Supporting Electronic Commerce

Through the application of concurrent engineering techniques using electronic commerce in conjunction with robust product data exchange standards, design organizations will be able to quickly and inexpensively locate an appropriate manufacturing site for their application-specific signal processor products. By reducing cost and time-to-market, the Manufacturing Interface contributes significantly towards the accomplishment of the RASSP program's goals.

5.0 Network Strategy

It is the goal of RASSP Enterprise infrastructure to implement tools which expand other associated capabilities of RASSP without compromising data and by protecting privacy of communcations. This takes several forms. It includes the addition of a secure communcations layer on top of World Wide Web transactions. This function provides for encrypted client/server sessions. Thus, using the WWW HyperText Transfer Protocol (http) as a transfer medium, RASSP related information may be hosted on a secure server. By combining with already existing http authentication schemes, confidential design information can be made available to selected users which is protected against unauthorized access.

1. Netscape Enterprise Server

The specific tool used to implement secure WWW sessions is Netscape's Commerce/Enterprise Server. This server uses Secure Sockets Layer (SSL) protocol to encrypt client/server sessions. It uses Public Key encryption technology. Encryption keys are exchanged by server and client when a session is initiated. A one-time session key is also used to ensure uniqueness during an individual session. The encryption session is performed transparently by client and server with no overt action required by the user of the client browser. Thus, in order for the sessions to be "private", authentication is built on top by means of tailored HypterText Markup Language (HTML) coding. This involves the use of usernames and passwords, whose exchange is automatically encrypted.

2. Viacrypt's Pretty Good Privacy (PGP)

Another component of secure communications is the implementation of the use of Pretty Good Privacy (PGP) email and data encryption schemes. This function again uses Public Key encryption technology along with a simple GUI to allow users to ensure that their messages can only be read by intended recipients. For data, it can be used to encrypt data files prior to their staging on a network server. Again, this ensures that confidential data is protected from unauthorized access.

Note that PGP comes in two varieties: freeware version and commercial version. The freeware version is available to individuals and the commercial version is for use by businesses. These versions are fully interoperable with one another.

5.1 Collaboration Tools

Collaboration is a term which is used to describe the interaction between participants in the RASSP design and development process. Specific tools are sought which improve the efficiency and quality of information exchange between individuals, while at the same time providing a communications platform used in conjunction with other on-going RASSP initiatives.

Two collaboration tools have been identified and implemented. These can be used with other RASSP Enterprise capabilities, e.g. the Manufacturing Interface function. The collaboration tools used, Communique and Cooltalk by Insoft, support a variety of exchange mechanisms. Included are a series of "tools". The tools are: Chat Tool, Audio Tool, Whiteboard Tool, TV Tool, Information Exchange Tool, and Image Tool. The only substantive difference between Communique and Cooltalk is that Communique includes the TV Tool and Cooltalk does not. In addition, Communique is a standalone product whereas Cooltalk has been bundled with the new version of the Netscape Navigator web browser. Any subset of these "tools" can be used in a specific application. In some cases the tools employed may not include the complete suite due to network firewall limitations. In particular, both the Audio and TV Tools are User Datagram Protocol (UDP) based, which are normally screened by corporate firewall implementations. The remaining tools are Transmission Control Protocol/Internet Protocol (TCP/IP) based and more likely to be supported. Nonetheless, any one of the aforementioned tools provides a significant enhancement of the information exchange capabilities between participants in the RASSP workflow process. A brief description of these tools follows.

A Communique Conference is first established by one individual. Other individuals are in turn invited to join the conference. Some initial coordination must first be established to ensure that each user has his or her Communique software executing, as this is a pre-requisite. The conference initiator issues invitations to others who in turn accept these invitations. In this manner the conference is established where any or all participants can then invoke specific tools. Each tool invoked is accessible or visible to each conference attendee. For example, if a chat session is initiated, each participant has an opportunity to view the chat exchange and join in, if desired.

The chat tool generate a viewable window which displays typed text by each participant, with a username preceding each entry. The audio tool provides the ability to conduct an audio teleconference and requires the use of microphones and speakers at each participating workstation. The whiteboard tool enables the use and display of a shared whiteboard application where each participant to generate graphic symbols, text, freehand sketchings, and import graphics. This tool is extremely valuable when used in the context of evaluating

detailed design material. The TV Tool is the video equivalent of the audio tool and requires that the workstation(s) is equipped with a camera. Setup parameters can be adjusted to configure the degree of audio/video synchronization. When this function is working properly over communications links possessing sufficient bandwidth it is as if the individuals are collaborating in physical proximity. The information exchange tool allows users to exchange and view data contained on floppy disks or CDs.

6.0 System Environment

This section identifies the infrastructure and design tools, and hardware configuration for the Build 2 version of the RASSP Enterprise System.

6.1 Infrastructure and Design Tools

This section identifies the infrastructure and design tools that form the Build 2 Enterprise System. The tools are listed in either a 3 or 4-column table. The first column contains the name of the tool. The second column indicates if the tool's architecture is client-server. The third column indicates the operating system of the host machine. If the tool has a client-server architecture, then the third column indicates the name of the server platform. The fourth column, if present, indicates whether the tool has been encapsulated into the Enterprise Framework. Most of the design tools were encapsulated for Build 1A in support of the Benchmark 3 program, but there have been some additions and updates of tool versions. Section 7.2, Hardware Configuration, identifies the exact version of each Build 2 tool.

6.1.1 Infrastructure Tools

Table 1, shown below, contains the infrastructure tool list. The infrastructure tools are used to develop the Enterprise Framework.

Tool Name	Client-Server	Platform
DM2	Yes	Solaris
DMM	Yes	SunOS

Table 1. Infrastructure Tool List

6.1.2 System Definition Tools

Table 2, shown below, contains a list of design tools used in the Systems Definition phase of the RASSP design methodology. The Build 2 version of the RASSP Enterprise System does not contain any systems definition workflows, so these tools have not yet been encapsulated into the framework. The exceptions are the Aspect and Interleaf tools. They have been encapsulated because they are used in other phases of the RASSP design methodology.

Tool Name	Client-Server	Platform	Encapsulated
Alta BONeS	Yes	SunOS	No
Alta SPW	Yes	SunOS	No
Ascent Logic RDD-100	No	SunOS	No
Aspect Explore CIS	Yes	Solaris	Yes
Interleaf TPS	Yes	SunOS	Yes
Lockheed Martin PRICE	Yes	SunOS / PC	No
Marconi RTM	Yes	SunOS	No
Mathworks Matlab	No	SunOS	No
MGC DSS	Yes	SunOS	No
MSI RAM/ILS	Yes	SunOS	No

Table 2. System Definition Tool List

6.1.3 Architecture Definition Tools

Table 3, shown below, contains a list of design tools used in the Architecture Definition phase of the RASSP design methodology.

Tool Name	Client-Server	Platform	Encapsulated
Alta BONeS	Yes	SunOS	No
Alta SPW	Yes	SunOS	No
Ascent Logic RDD-100	No	SunOS	No
Aspect Explore CIS	Yes	Solaris	Yes
MATRIXx	Yes	SunOS	No
Interleaf TPS	Yes	SunOS	Yes
JRS Architecture Definition	Yes	SunOS	Yes
JRS Assignment	Yes	SunOS	Yes
JRS Graph Development	Yes	SunOS	Yes
JRS VHDL	Yes	SunOS	Yes
Lockheed Martin PRICE	Yes	SunOS / PC	No
Marconi RTM	Yes	SunOS	No
Mathworks Matlab	No	SunOS	No
MCCI	No	SunOS	No
MGC AutoTherm	No	SunOS	Yes
MGC DA-LMS	No	SunOS	Yes
MGC QuickVHDL	No	SunOS	Yes
MGC VTM:TOP	No	SunOS	Yes
MSI RAM/ILS	Yes	SunOS	No
PGSE	No	SunOS	No
Summit Visual HDL	No	SunOS	Yes

Table 3. Architecture Definition Tool List

6.1.4 Detailed Design Tools

Table 4, shown below, contains a list of design tools used in the Detailed Design phase of the RASSP design methodology.

Tool Name	Client-Server	Platform	Encapsulated
Aspect Explore CIS	Yes	Solaris	Yes
Insoft Communique	No	SunOS	Yes
Interleaf TPS	Yes	SunOS	Yes
MGC AutoTherm	No	SunOS	Yes
MGC DA-LMS	No	SunOS	Yes
MGC FabLink	No	SunOS	Yes
MGC Layout	No	SunOS	Yes
MGC PTM Site	No	SunOS	Yes
MGC QuickVHDL	No	SunOS	Yes
MGC VHDLWrite	No	SunOS	Yes
MGC VTM:TOP	No	SunOS	Yes
NeoCAD	No	SunOS	Yes
Netscape 3.0	No	SunOS	Yes
SCRA AP210 Translator	No	SunOS	Yes
SCRA Producibility Advisor	No	SunOS	Yes
Summit TDS	No	SunOS	No
Summit Visual HDL	No	SunOS	Yes
Synopsys Design Compiler	No	SunOS	Yes
Teradyne Victory	No	SunOS	Yes
TI Asset	No	PC	No
TSTB/Waves	No	SunOS	Yes
VHDL Cover		SunOS	No
Xilinx		SunOS	No

Table 4. Detailed Design Tool List

6.2 Hardware Configuration

Node Name	User	Type	Op. Sys.	IP Address
HICKS	efuser		SunOS	
VASQUEZ	efadmin		Solaris	
		TD2	Windows NT	

Table 5. Hardware Configuration

Appendices

A.1 Design Tool Encapsulation Guide

This appendix describes the procedure for encapsulating design tools into the Build 2 version of the RASSP Enterprise Framework. It does not address the installation of the individual design tools.

Encapsulation Procedure

The encapsulation procedure basically consists of two steps. First the design tools are declared and then they are instantiated for a particular project. Design tools are declared within each task of a workflow, and instantiated by creating a tool definition file. A tool definition file is an ASCII file with a *.tol* extension.

Tool Declarations

Design tools are declared for each workflow process. A workflow represents the implementation of a design process (e.g. module design, ASIC design, etc.). For RASSP, workflows were developed using Intergraph's DMM (Design Methodology Manager) tool. The DMM Builder provides a GUI (Graphical User Interface) for constructing workflows. Figure 11 shows a portion of a RASSP workflow constructed using DMM Builder.

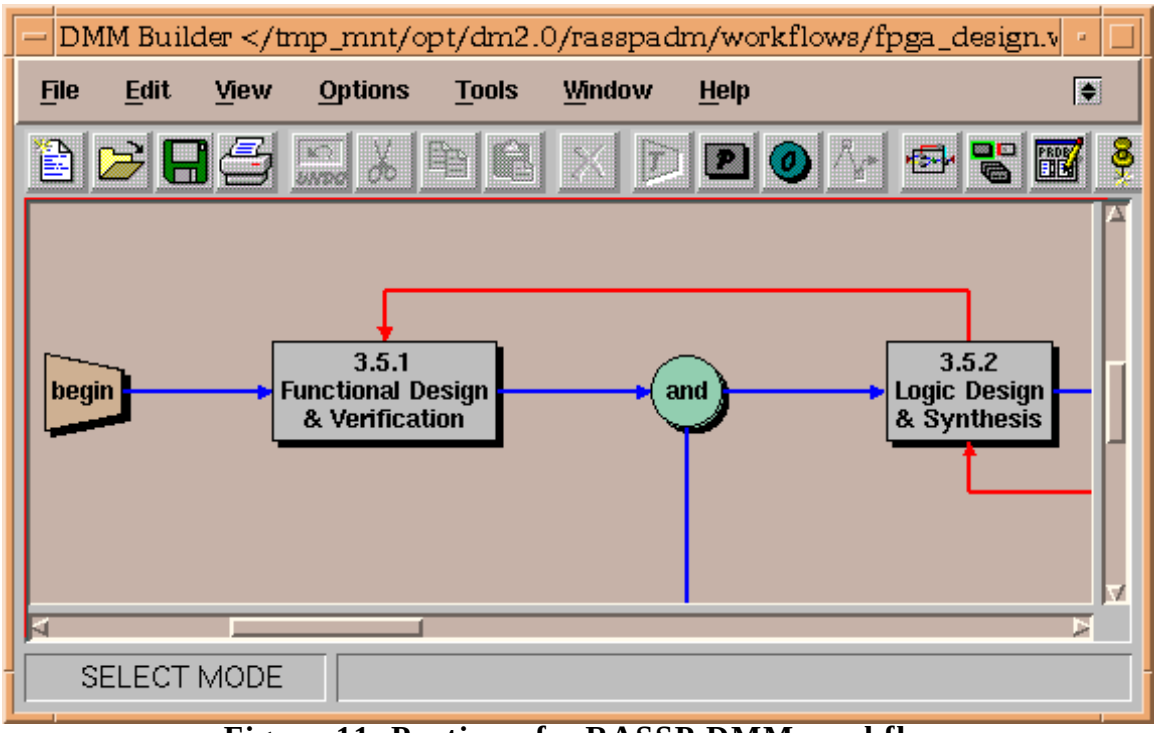

Figure 11. Portion of a RASSP DMM workflow

Design tools are declared in the attributes folder of the definition palette for each workflow process at the time the workflow is constructed in DMM. To access the attributes folder, within DMM Builder, simply double-click on a workflow process with the left mouse button. When the process definition palette is displayed, select the attributes folder. Figure 12 shows the attributes folder for workflow process 3.5.1 from figure 11.

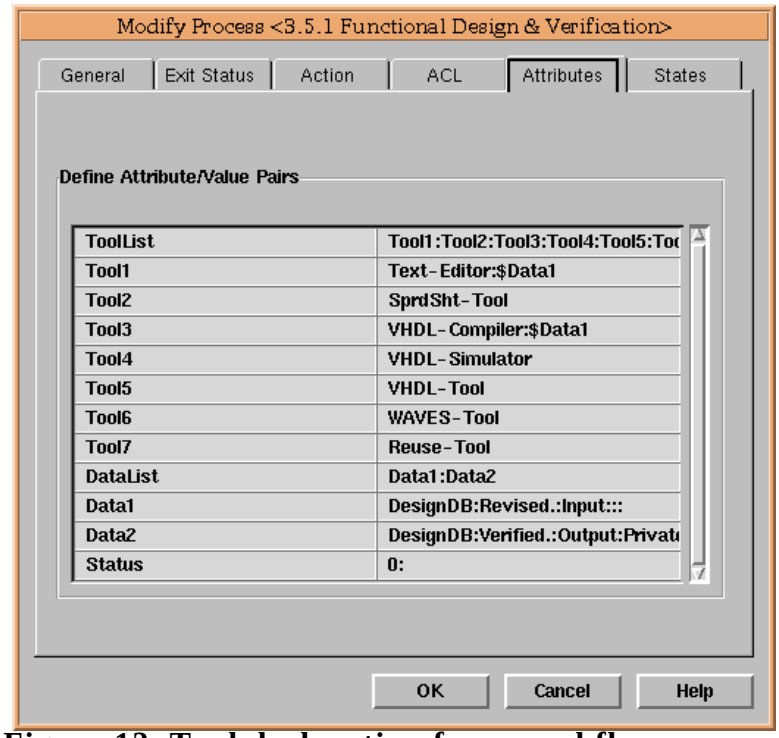

Figure 12. Tool declaration for a workflow process

The attributes folder contains attribute/value pairs. An attribute named *ToolList* is used to specify a list of tools that will be invoked within that process. It's value is a list of attribute names separated by colons.

ToolList Tool1:Tool2:...:Tooln

where *Tool1, Tool2,...,Tooln* are themselves attribute names which will be defined in separate lines. The value for each *Tool1, Tool2,...,Tooln* attribute is also a list of fields separated by colons. These fields are defined as follows:

Tool1 generic tool name[:$<arg 1>:$<arg 2>:...:$<arg n>]

where *generic tool name* is the declaration of a design tool. This field may not contain spaces or underscores. *[:$<arg 1>:$<arg 2>:...:$<arg n>]* are optional fields which would contain command line arguments that are passed to the tool upon invocation. These command line arguments would most likely be input and output files which are attached to business items. More than one argument may be specified, but it must be separated by a colon ":". There is no upper limit on the number of design tools that can be declared for each workflow process.

Another attribute named *DataList* is used to specify the DM2 business items that have been defined as inputs to, outputs from, and controls for, the workflow process. This attribute will not be discussed in this document. The last line which contains the attribute/value pair

Status 0:

is important and is required for all workflow processes. When the Attributes folder is completed, click on "OK" to close and save it.

Figure 12 shows seven design tools declared for workflow process 3.5.1. They are: *Text-Editor, SprdSht-Tool, VHDL-Compiler, VHDL-Simulator, VHDL-Tool, WAVES-Tool,* and *Reuse-Tool*. These are generic names for the design tools which will be instantiated with specific tool names from the tool definition files. Design tool declarations can be added and deleted from workflow processes after the workflow has been developed by editing the workflow file within the DMM Builder.

Tool Instantiations

Every design tool that is declared in the DMM workflows must be instantiated. Instantiation is accomplished by creating a separate tool definition file for each design tool. The tool definition file must have a *".tol"* extension. The tool definition files are all stored in a single directory that is defined when the RASSP environment is installed The syntax for the tool definition file is as follows

Tool File Format 2.0.0
<file name>:
$TOOL_ARG_LIST
<arg 1>:<arg value>
<arg 2>:<arg value>
.
.
<arg n>:<arg value>
$END
$TOOL_OPT_LIST
<opt name>:<value>:<required>:<default>:<prompt>:<description>
$END
<toolpad name>:<tool name>:<working dir>:<icon file>:<cmd argument>
$TOOL_ENV_LIST
<env 1>:<env value>
<env 2>:<env value>
.
.
<env n>:<env value>
$END

All lines start in the first column of the file. Fields are separated by colons ":", therefore no colons are permitted in any of the fields. Fields that are enclosed by "<>" are fields that are replaced with tool specific information. Spaces are not permitted in the following fields: *<arg 1..n>*, *<opt name>*, *<value>*, *<required>*, *<default>*, *<tool name>*, *<working dir>*, *<icon file>*, and *<env 1..n>*. Spaces are permitted in the *<arg value>*, *<prompt>*, *<description>*, *<toolpad name>*, *<cmd argument>*, and *<env value>* fields.

The *<file name>* field contains the base name of the tool definition file without the extension (e.g. VHDL-Tool:). The trailing colon is required for this field.

The $TOOL_ARG_LIST section is where command line arguments for the tools are defined. The values of the arguments will be passed in, in the order they are specified, when the tool is invoked. Arguments in this section can be referenced in the *<toolpad name>*, *<cmd argument>*, and *<env value>* fields, as will be explained in the next section. The end of this section is indicated by the $END line.

The $TOOL_OPT_LIST section is where all optional arguments are defined. Optional arguments are usually switches that can be set in the command line when invoking the tool

(e.g. qvcom -nodebug <vhdl file>). A description of the fields in this section is as follows: The <*opt name*> field is the string name of the optional argument. The <*value*> field contains the argument's value. The <*required*> field indicates if this argument is required. If this field contains the string "required", then the argument is required. Otherwise it's optional. The <*default*> field is not currently used and should be left blank. The <*prompt*> field defines the label that can be displayed in the GUI. The <*description*> field contains a descrition of the optional argument. The optional arguments defined in this section are referenced in the <*cmd argument*> field, as will be explained in the next section. The end of this section is indicated by the $END line.

The <*toolpad name*> field is where the name that appears on the DMM toolpad is defined. The <*tool name*> field is the executable path name. Specifying the full path is not necessary if the path is contained in the user's default PATH setting.

The <*working dir*> field specifies the working directory for the tool. The <*icon file*> field contains the name of the file containing the tool icon that is also displayed on the DMM toolpad. The <*cmd argument*> field contains the command line arguments that will be passed to the tool when it is invoked.

The $TOOL_ENV_LIST section contains the settings of any special UNIX environment variables that are needed by the tool. These environment variables are set just before the tool is invoked and remain in existence during execution of the tool. Once the tool is exited, the environment variable settings are removed.

Examples

The example files in this section are actual tool definition files for design tools that are currently encapsulated within the Build 2 version of the RASSP Enterprise Framework. These examples also correspond to the tools declared for workflow process 3.5.1 from figure 11.

```
Tool File Format 2.0.0
Text-Editor:
$TOOL_ARG_LIST
iFile
$END
$TOOL_OPT_LIST
$END
Emacs editor :emacs::emacs.bmp:$iFile
$TOOL_ENV_LIST
$END
```

Example 1. Text-Editor.tol

Example 1 contains the "Text-Editor.tol" file which instantiates the emacs editor. The <*file name*> field contains the base name of the tool definition file which in this case is "Text-Editor". The $TOOL_ARG_LIST section contains one argument "iFile" which represents the input file to the emacs editor. It's value is passed in from the attributes folder of the definition palette for the DMM workflow process as indicated in Figure 2. The $TOOL_OPT_LIST section is empty since there are no additional arguments to be defined for the emacs editor. The <*toolpad name*> field contains "Emacs editor" which is displayed inside the DMM toolpad when the workflow process is executed. The <*tool name*> field contains the executable name "emacs" for invoking the tool. The <*working dir*> field is not utilized for this tool and is left blank. That is why there are two colons "::" between the

<tool name> and *<icon file>* fields. The *<icon file>* field contains "emacs.bmp" which is the name of the file containing the icon that also gets displayed inside the DMM toolpad when the workflow process is executed. The *<cmd argument>* field contains only one value, "$iFile", which is passed into the emacs editor as a command line argument. This argument contains the name of the file that the editor is invoked on. There are no UNIX environment variables defined for this tool, so the $TOOL_ENV_LIST section is empty.

```
Tool File Format 2.0.0
VHDL-Tool:
$TOOL_ARG_LIST
$END
$TOOL_OPT_LIST
$END
Summit VisualHDL:visual_hdl::vishdl.bmp
$TOOL_ENV_LIST
VISUALHDL:$HOME
$END
```

Example 2. VHDL-Tool.tol

Example 2 contains the "VHDL-Tool.tol" file which instantiates the Summit Visual HDL design tool. The *<file name>* field contains the base name of the tool definition file which in this case is "VHDL-Tool". The $TOOL_ARG_LIST section does not contain any arguments, so it is left blank. Similarly the $TOOL_OPT_LIST section is empty because there are no addition arguments to be defined for the Summit Visual HDL tool. The *<toolpad name>* field contains "Summit VisualHDL" which is displayed inside the DMM toolpad when the workflow process is executed. The *<tool name>* field contains the executable name "visual_hdl" for invoking the tool. The *<working dir>* field is not utilized for this tool and is left blank. That is why there are two colons "::" between the *<tool name>* and *<icon file>* fields. The *<icon file>* field contains "vishdl.bmp" which is the name of the file containing the icon that also gets displayed inside the DMM toolpad when the workflow process is executed. Since the $TOOL_ARG_LIST section is empty, the *<cmd argument>* field is also left empty. The $TOOL_ENV_LIST section contains the UNIX environment variable "VISUALHDL" which is set to the user's home directory when the tool is invoked.

```
Tool File Format 2.0.0
VHDL-Compiler:
$TOOL_ARG_LIST
iFile
$END
$TOOL_OPT_LIST
dSwitch:-nodebug::::optional debug switch
$END
MGC QuickVHDL Compiler:new_qvcom:$HOME/$USER.wl:qvcom.bmp:$dSwitch
$iFile
$TOOL_ENV_LIST
QUICKVHDL:$HOME/quickvhdl.ini
$END
```

Example 3. VHDL-Compiler.tol

Example 3 contains the "VHDL-Compiler.tol" file which instantiates the Mentor Graphics QuickVHDL compiler. The *<file name>* field contains the base name of the tool definition

file which in this case is “VHDL-Compiler”. The $TOOL_ARG_LIST section contains one argument “iFile”. It’s value is passed in from the attributes folder of the definition palette for the DMM workflow process as indicated in Figure 2. The $TOOL_OPT_LIST section contains the optional argument “dSwitch” which represents the optional nodebug switch that can be set for this tool. The <*value*> field contains “-nodebug” which is the value for this switch. The <*required*>, <*default*>, and <*prompt*> fields are all left blank. The <*description*> field contains “optional debug switch” as a brief description of this switch. The <*toolpad name*> field contains “MGC QuickVHDL Compiler” which is displayed inside the DMM toolpad when the workflow process is executed. The <*tool name*> field contains the name “new_qvcom”. This is the name of the UNIX shell script which invokes the tool. The <*working dir*> field contains the tool’s UNIX working directory which is set to “$HOME/$USER.wl”. The <*icon file*> field contains “qvcom.bmp” which is the name of the file containing the icon that also gets displayed inside the DMM toolpad when the workflow process is executed. The <*cmd argument*> field contains two values “$dSwitch” & “$ifile” which are passed into the Mentor Graphics QuickVHDL compiler as command line arguments. There are no UNIX environment variables defined, so the $TOOL_ENV_LIST section is empty.

Design Tool Invocation

This section describes how design tools are invoked once they have been encapsulated into the DMM workflows. When the RASSP system is executed, the DMM displayer will display the project workflows. Workflow tasks which are startable may be executed by double-clicking on it. Once the workflow task has been started, DMM will display a tool launch pad which contains the tool names and icons of the tools that were declared and instantiated for that particular process. Figure 13 shows an example of the DMM toolpad from which design tools are invoked.

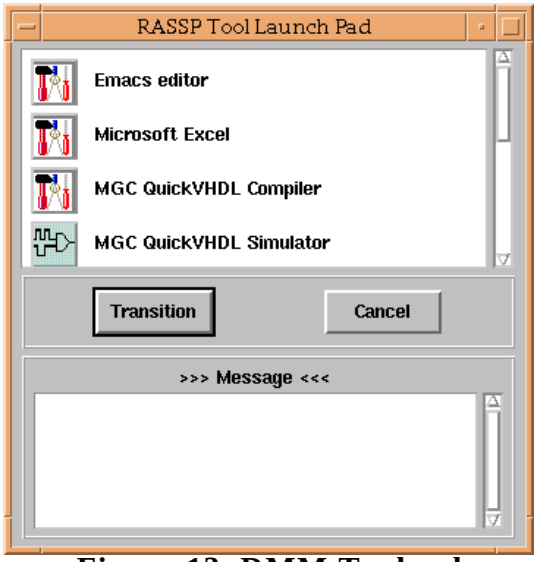

Figure 13. DMM Toolpad

The tool names and icons that appear in the DMM toolpad are all defined in the tool attribute files. To invoke a tool, the user simply clicks the icon next to the tool name. The design tool is then invoked.