Rapid Prototyping of Application-Specific Signal Processors (RASSP)

The Configuration Management Model for the RASSP System Version 3

Generated By: B. Kalathil (July 1994)
Modified By: B. Kalathil (September 1994 - Version 2)
Modified By: D. Blanchard (August 1996 - Version 3)

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

Date:
June 23, 1996

Table Of Contents

1 Introduction

2 CM Process Example

3 Configuration Management in the RASSP System

3.1 Shared and Private Workspaces

3.2 Data Object Versioning

4 RASSP CM Mechanics

4.1 Introduction

4.2 Workspace Functions

- **4.2.1** Creating a Workspace
- **4.2.2** Accessing an arbitrary workspace
- **4.2.3** Accessing the Child Workspace
- **4.2.4** Accessing the Parent Workspace
- **4.2.5** Making a workspace visible

4.3 Version Management Functions

- **4.3.1** Creating a Configuration
- **4.3.2** Inserting Data objects into a configuration
- **4.3.3** Checkout
- **4.3.4** Checkin
- **4.3.5** Accessing Child Versions
- **4.3.6** Accessing Parent Version
- **4.3.7** Naming Versions
- **4.3.8** Retrieving a named version

5 Implementation Strategy

The Configuration Management Model for the RASSP System

1 Introduction

The Rapid Prototyping of Application-Specific Signal Processors (RASSP) is an Advanced Research Projects Agency (ARPA)/Tri-Service program aimed at dramatically improving the process of design, manufacture, test and procurement of digital signal processors. The RASSP program will deliver an integrated system called the RASSP system, which integrates the CAD tools used in the RASSP design process under a framework referred to as the enterprise framework. An *enterprise framework* provides the facilities and services necessary to integrate the automated processes of an enterprise. In the RASSP system the enterprise framework provides support for workflow management, design data management, library management, computer-supported collaborative work and remote tool access. The workflow management subsystem of the RASSP enterprise system enables a RASSP system administrator to model and enforce a particular design methodology for a project. The data management subsystem of the enterprise framework provides facilities for configuration managing, and controlling access to design data files that may reside at various sites in a

computer network. The library management subsystem provides facilities for cataloging, classifying and storing reusable design components; as well as mechanisms for searching for reusable components.

Configuration Management (CM) in the RASSP 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. Different CAD vendor tools provide different mechanisms to support the configuration management process. A *mechanism* is an individual function of a system. The mechanisms provided by the tools not only vary in scope, but also in their semantics. In an integrated product development environment which involves several vendor tools, such as RASSP, diverse and incompatible configuration management mechanisms across tools can lead to the inefficiencies in the design process listed below:

- There is no common way of handling the configuration management of a product throughout its life cycle
- The design engineers working on a project have to learn several different paradigms of CM
- The CM data on a product generated by one tool cannot be used by another tool

We propose in this document a common model of configuration management that may be adopted by the RASSP enterprise framework tools and the CAD tools. This model 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. The model does not cater to the version management of reusable library components that are created outside of the design process. The release management of reusable library components is described in [Martin Marietta, 1994]. We have specified a common minimal set of configuration management mechanisms that need to be provided by the tools to support the proposed CM model. To facilitate the exchange of CM data between tools the CM data generated by each tool will be modeled using the CM conformance class of the STEP (Standard for the Exchange of Product Model Data) standard AP203 [ISO, 1993]. An important criterion we have followed in the development of the model and the mechanisms is that they should be generic enough to allow an organization to adopt any CM process it chooses to. An example CM process is described in section 2.

The RASSP enterprise framework tools that create and manipulate data during the process of design include file managers such as Intergraph's Electronic Desktop Manager (EDM), Intergraph's Network File Manager (NFM); STEP database tools, and CAD frameworks such as the Mentor Falcon Framework. The CAD tools include tools used in the various stages of the RASSP design process, such as system design, architecture design, simulation and hardware design. Examples include tools such as RTM, RDD100, Matlab, SPW, JRS, GEDAE, Ptolemy, Omniview, Synopsis and Mentor hardware design tools.

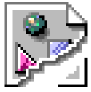
Apart from the managing of versions of design objects, the CM process of an organization includes generating reports such as problem reports, engineering change proposals, specification change notices and notices of revision. The generation and management of these reports will be done external to the CAD systems, using a document management system such as Interleaf [Interleaf, 1993].

In the next section we describe an example CM process in an organization. In section 3 we propose a common model of configuration management for the RASSP system. In section 4 we describe a common set of CM mechanisms needed for supporting the proposed CM model. We discuss the implementation strategy and propose a schedule for implementation in section 5.

2 An Example CM Process

Figure 1 shows an example CM process adopted by an organization. We have used an IDEF3 notation to represent the workflow. The boxes represent individual activities in a process, and the links between the activities represent precedence relationships between activities. The links are annotated with information about the data that flows between two activities -- the state of the data and the type of data separated by a '*'. A junction box, represented by a box with a vertical line parallel to the left edge and an 'X' within the box, is

used to model alternate paths within a workflow. The arrows coming into the bottom edge of an activity box indicate the mechanisms that are involved in the activity, usually the job classification of the individual(s) performing the activity.



3. Configuration Management in the RASSP System

3.1 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]. Workspaces are organized in a hierarchical fashion as shown in figure 2, with a global workspace at the root of the hierarchy, shared workspaces as the intermediate nodes in the hierarchy, and private workspaces as the leaves in the hierarchy. The links in a workspace hierarchy represent a parent-child relationship between the linked workspaces. For example, in figure 2 the workspace "Private WS2" is a child of the workspace "Shared WS1" (and the workspace "Shared WS1" is the parent of the workspace "Private WS2").

Workspaces provide for varying levels of sharing of data objects. A user of a workspace has visibility to all the objects residing in the workspace and the objects residing in the ancestor workspaces of the workspace. Thus all users of the database have visibility to data objects residing in the global workspace.

3.2 Data Object Versioning

We propose a data object versioning scheme where related data objects that evolve at the same time are

grouped together as configurations, and versioning is managed at the level of configurations. New configuration objects are typically created in a private workspace, at which point the configuration is considered a transient version of the configuration. A transient version may be updated or deleted. Once the transient version of a configuration reaches a state of maturity suitable for sharing with other designers in a project, it is promoted to a working version of the configuration, by checking in the configuration from the private workspace where it resides, to its parent workspace. A working version may not be updated but may be deleted. Working versions of configurations that are considered to represent the final state of design are promoted to released versions by checking in them to the global workspace. A released version may not be updated nor deleted. We use the notation statei > statej to denote that statei is a higher state that statej. Thus released > working > transient.

The operations Baseline and Revise are special cases of checkin and checkout applied to the Global workspace.

Workspace	State	Read	Delete	Update	Approve	Baseline	Revise
Global	Released	Yes	No	No	No	No	Yes
Shared	Approved	Yes	Yes	No	No	Yes	No
	Working	Yes	Yes	No	Yes	No	No
Private	Transient	Yes	Yes	Yes	No	No	No

Figure 3. Permissible operations for workspaces

A new transient version c_j of a configuration c, may be created by checking out an existing version c_i residing in a workspace w_m to a workspace w_n that is a direct or indirect descendent of w_m. The source version c_i may be in one of the three states prior to checkout -- released, working or transient. If the state of c_i prior to checkout is released or working, the state of c_i remains unchanged after checkout. However, if the state of c_i is transient prior to the checkout, c_i is promoted by the system to a working version as part of the checkout operation.

The versions of a configuration are organized as a directed acyclic graph (DAG), as shown in figure 4, and is commonly referred to as a version tree. The new nodes in the tree start out as a transient version and progressively become working versions and then released versions. A version in a version tree may be deleted only if it is a transient or a working version, and it is a leaf node in the tree. A directed link between two versions i and j in a version tree represents the *is-derived-from* relationship, i.e., version j is derived from version i. Also, version i is said to be the *parent* of version j, and version j is said to be a *child* of version i. We use the notation c_i -> c_j to represent the parent-child relationship between the two versions i and j of the configuration c.

The following rule applies to a version tree:
VT-Rule1: Given two versions c_i and c_j of a configuration c, such that c_i -> c_j, then the state of c_j should be less than or equal to the state of c_i.

4. RASSP CM Mechanisms

4.1 Introduction

We propose in this section a minimal set of CM mechanisms that needs to be supported across the RASSP enterprise framework tools. This set will provide the basis for configuration management of design data in the different phases of the RASSP design process. The set of CM mechanisms proposed here is intended to serve as a common minimal set; an individual CAD tool may support more than the minimal set proposed here.

We categorize the CM mechanisms into two classes -- workspace functions and version management functions. In the following sections we use a C-like pseudo code to describe the functions. For example,

```
Bar my_func (Foo a_foo);
```

describes a function "my_func" that takes as a parameter an object of type "Foo" and returns an object of type "Bar". We use names starting with capital letters, such as "Workspace" to denote the type of object, and names starting with small letters, such as "create_workspace" and "parent_workspace" to denote a function name or a parameter name.

```
Bar another_func (foo *a_foo=0);
```

describes a function "another_func" that takes as a parameter a pointer (denoted by the *) to an object of type "foo". The "= 0" is a default value for the parameter if one is not provided. Thus the parameter "a_foo" is an optional parameter for the function "another_func", while it is a required parameter for the function "my_func".

The C-like style we are using to describe the mechanisms is for the brevity of the descriptions, and does not have any implications as to the implementations of these functions, nor the user interface provided by the

systems to these functions.

4.2 Workspace Functions

4.2.1 Creating a workspace

```
Workspace *create_workspace (char *workspace_name,
Workspace *parent_workspace=0);
```

Creates a child workspace of the specified parent workspace, and assigns it the supplied name. If a parent workspace is not specified, the new workspace is made a child of the global workspace. The global workspace is a system-created workspace that exists in every database, and has the name "global_workspace" assigned to it.

4.2.2 Accessing an arbitrary workspace

```
Workspace *get_workspace (char *workspace_name);
```

Returns a pointer to the workspace with the specified name.

4.2.3 Accessing child workspaces

```
Workspace_List child_workspaces (Workspace *a_workspace);
```

Returns a list of pointers to child workspaces of the specified workspace.

4.2.4 Accessing the parent workspace

```
Workspace *parent_workspace (Workspace *a_workspace);
```

Returns a pointer to the parent workspace of the specified workspace.

4.2.5 Making a workspace visible

```
void set_current_workspace (Workspace *a_workspace);
```

Sets the specified workspace to be the current workspace for the application program. The application program can access only objects that reside in the current workspace, or in the ancestor workspaces of the current workspace.

4.3 Version Management Functions

4.3.1 Creating a configuration

```
Configuration *create_configuration ();
```

Creates a new configuration and returns a pointer to it.

4.3.2 Inserting data objects into a configuration

```
void insert_into_configuration (Configuration *a_configuration,
void *a_design_object);
```

Inserts the design object pointed to by "a_design_object" into the specified configuration.

4.3.3 Checkout

```
Configuration *checkout (Configuration *a_configuration,
char *version_name=0);
```

Creates a new version of the configuration pointed to by "a_configuration", and makes the new version visible in the current workspace. The new version may also be provided an optional version name. The version name

is an arbitrary name provided by the user. If the configuration pointed to by a_configuration is a transient version, it is promoted by the system to a working version, by performing a checkin operation (see section 4.3.4), before performing the checkout. The configuration pointed to by "a_configuration" may reside in the current workspace, or in any of the ancestor workspaces of the current workspace. This function returns a pointer to the newly created version of the configuration.

4.3.4 Checkin

```
void checkin (Configuration *a_configuration);
```

Checks in the configuration pointed to by "a_configuration". This results in the configuration being made visible to the parent workspace of the current workspace. If the parent workspace is the global workspace, then the configuration is promoted to a released version, otherwise it is promoted to a working version. A working version of a configuration that is checked in to a non global workspace is not promoted in the process of checkin, but is made visible to the parent workspace.

4.3.5 Accessing child versions

```
Configuration_List child_versions
(Configuration *a_configuration);
```

Returns a list of pointers to the child versions of the configuration pointed to by a_configuration.

4.3.6 Accessing the parent version

```
Configuration *parent_version (Configuration *a_configuration);
```

Returns a pointer to the parent version of the configuration pointed to by "a_configuration".

4.3.7 Naming versions

```
void name_version(Configuration *a_configuration, char *a_name);
```

Assigns the character string pointed to by "a_name" as the name of the specified configuration.

4.3.8 Retrieving a named version

```
Configuration *get_named_version
(Configuration *a_configuration, char *a_name);
```

Returns the version of the specified configuration that has the name pointed to by "a_name". If no such version exists a null pointer is returned.

5 Implementation Strategy

The implementation of the CM mechanisms will be done by the individual tool vendors, both in the case of the enterprise framework tools and the CAD tools. In the case of CAD tools, the CM model will be implemented in one representative tool in each functional area of the RASSP design process, to demonstrate the value of the common CM model.

References

[Cattell, 1991] Cattell, R.G.G., *Object Data Management* , Massachusetts: Addison Wesley, 1991.

[Interleaf, 1993] Interleaf Inc., *Getting Started with Interleaf 6 for Motif* , Waltham, Massachusetts,