MARTIN MARIETTA LABORATORIES • MOORESTOWN

MARTIN MARIETTA

Rapid Prototyping of Application-Specific Signal Processors (RASSP)

The Authorization Model for the RASSP System

Version 2

Martin Marietta RASSP Team
Martin Marietta Laboratories • Moorestown
Bldg. 145
Moorestown Corporate Center
Moorestown, New Jersey 08057

Date:

September 9, 1994

The Authorization 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.

The various enterprise-level tools and CAD tools use diverse and incompatible models and mechanisms for authorization. Thus currently the authorization information cannot be shared among the various tools used in the RASSP enterprise framework. Authorizations have to be specified separately in the various tools and the management of the consistency of the authorization information among the tools in an enterprise is a management nightmare and a significant cost overhead. In this report we propose a common generic model of authorization that may be adopted by the RASSP enterprise framework tools and the CAD tools. We have specified a common minimal set of authorization management mechanisms that need to be provided by the tools to support the proposed model. To facilitate the exchange of authorization data between tools the authorization data generated by each tool will be modeled using the Configuration Management conformance class[1] of the STEP[2] (Standard for the Exchange of Product Model Data) standard AP203 [ISO, 1993]. Our model of authorization is based on the authorization model proposed for object data management systems by [Rabitti, 1991]. 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 authorization policy it chooses to. Example authorization policies are described in section 2.

In the next section we propose an authorization model for the RASSP system. We propose a set of mechanisms to support the RASSP authorization model in section 3. In section 4 we outline an implementation strategy and present a schedule for the implementation of the authorization model.

2 The RASSP 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 a authorization role in an authorization 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. Authorization objects in a database are organized as a directed acyclic graph as shown in figure 1. An *authorization role* is a collection of users that have the same set of authorizations on the same set of objects. The authorization roles in an organization are also organized as a directed acyclic graph as shown in figure 2. An *authorization type* is a type of operation that may be performed on a data object. The authorization types for a database are also organized as directed acyclic graphs as shown in figures 3 and 4. In figure 3 the "Grant" authorizations

[1]A conformance class of a STEP Application Protocol (AP) is a subset of the AP an application may support, but still be in compliance with the AP.
[2]STEP is the Standard for Exchange of Product Model Data. In RASSP, STEP standards are used primarily for exchanging product data among tools.

are authorizations to grant an authorization to another role in the role hierarchy. The authorization type hierarchy for projects also contains the "Grant" authorizations, but are not shown in figure 4 to avoid cluttering the figure. The directed links between two nodes in a hierarchy represent an *implication relationship* between the nodes. For example, an authorization for the engineering manager to update design data, implies the authorization to update system definition data, architecture data, etc., as they follow design data in the authorization object hierarchy. The authorization implies the same authorization for the project manager also, as the engineering manager follows project manager in the authorization role hierarchy. Also, since the operation read follows the operation update in the authorization type hierarchy for data objects, the authorization to update design data implies an authorization to read design data as well.

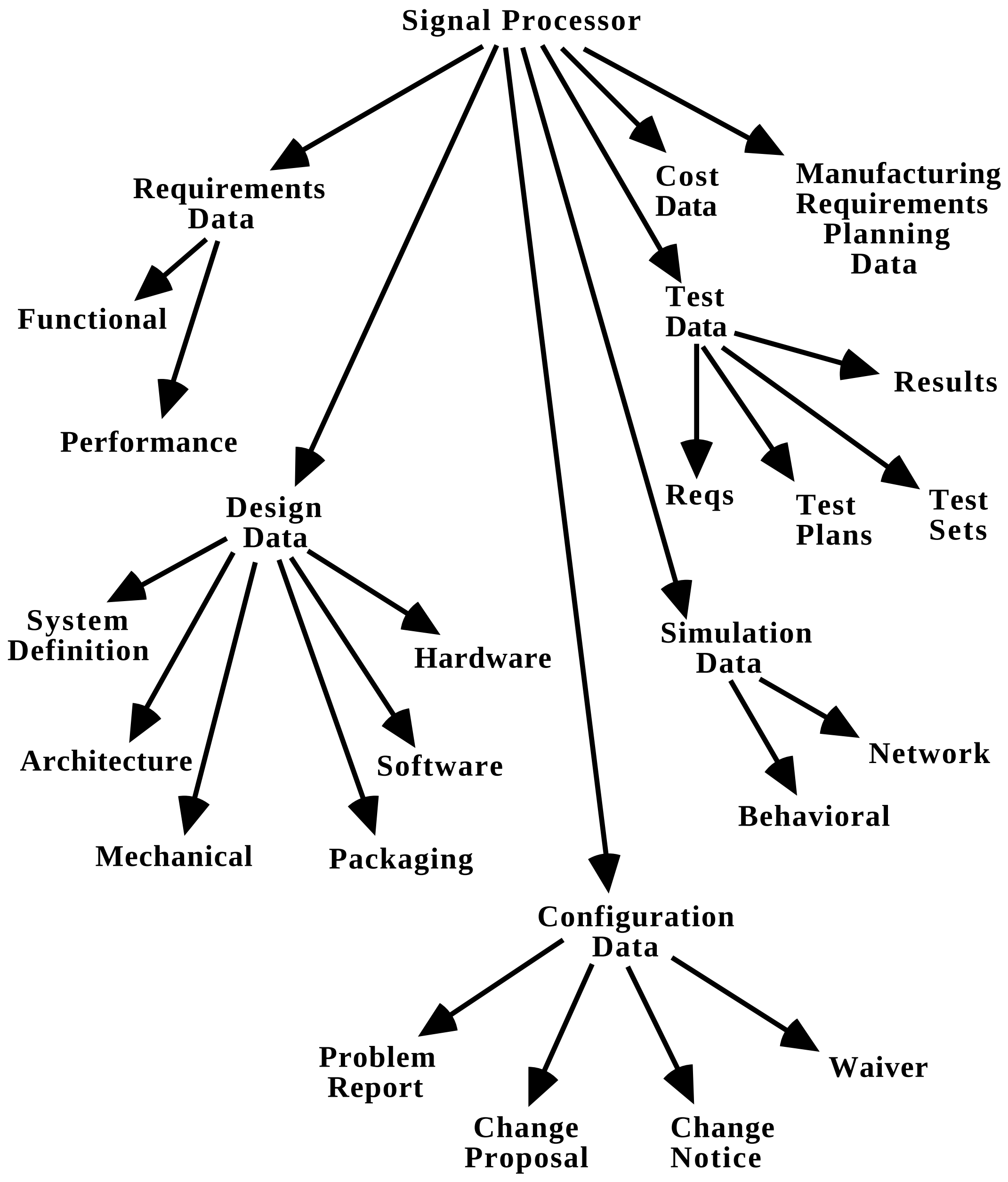

Figure 1. An example authorization object hierarchy

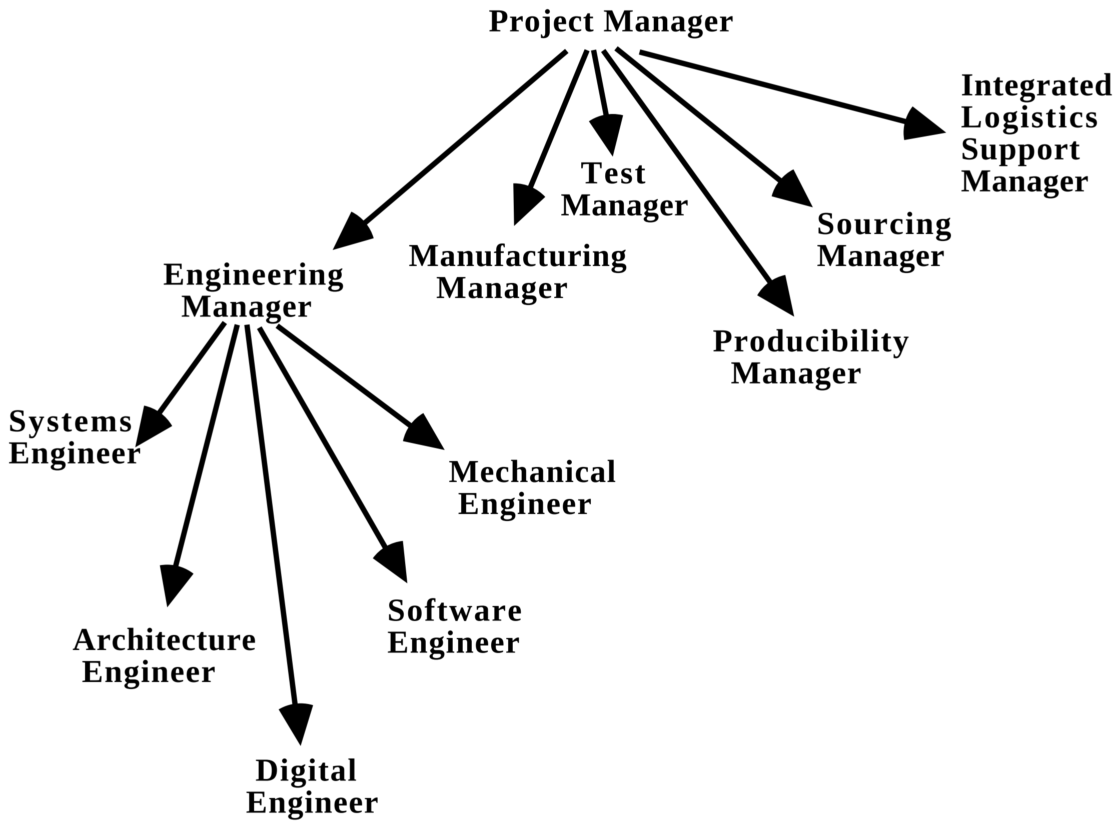

Figure 2. An example authorization role hierarchy

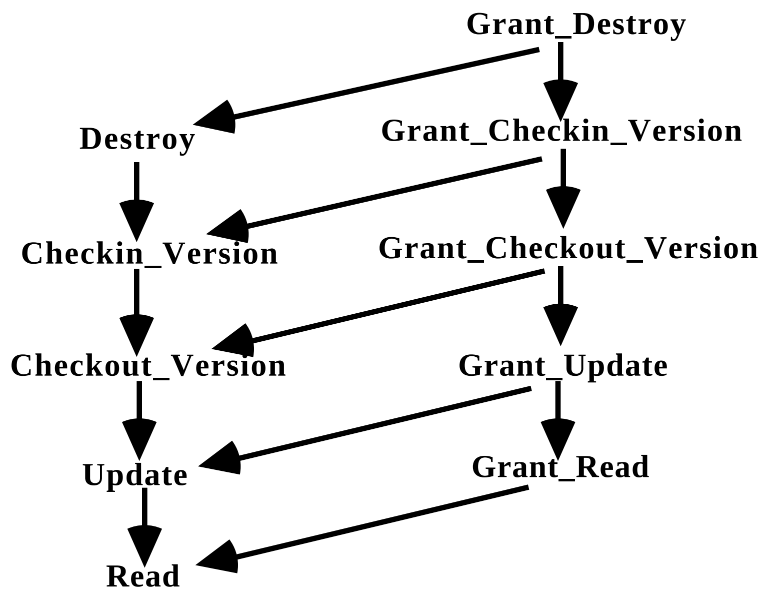

Figure 3. The authorization type hierarchy for design data objects

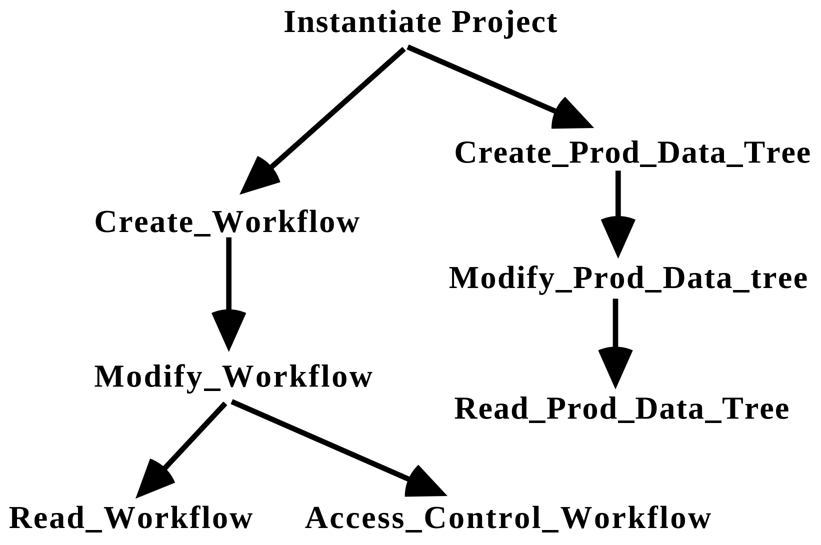

Figure 4. The authorization type hierarchy for projects

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 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. For example, a positive authorization for a particular role to update design data (in the example authorization object hierarchy in figure 1) may be overridden by a negative authorization for the same role to update mechanical design data. Similarly, a positive authorization for the engineering manager (in the example authorization role hierarchy in figure 2) to update configuration data (in the example authorization object hierarchy in figure 1) may be overridden by a negative authorization to update waiver data.

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. Each authorization type has an associated operation such as checkin, checkout, update etc. Thus, defining a new authorization type will typically involve adding a new functionality to the system.

3 Mechanisms to support the RASSP Authorization Model

We describe in this section four sets of mechanisms the data management subsystem of RASSP needs to provide in order to support the RASSP authorization model. In the following sections we use C-like functions to describe the mechanisms. 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 "Authorization_Object" to denote the type of object, and names starting with small letters, such as "create_authorization_object" and "parent" 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 mechanisms, nor the user interface provided by the systems to these mechanisms.

3.1 Mechanisms to manipulate the authorization object hierarchy

3.1.1 Creating an authorization object

```
Authorization_Object *create_authorization_object
                    (Authorization_Object *parent=0, char *name);
```

Creates a new authorization object, as a child of the specified parent authorization object and assigns it the specified name. If no parent authorization object is specified then the root node of a new authorization object hierarchy is created.

3.1.2 Deleting an authorization object

```
void delete_authorization_object
            (Authorization_Object *an_authorization_object);
```

Deletes the subset of the authorization object hierarchy rooted at the specified authorization object, from the authorization object hierarchy.

3.1.3 Adding a child to an authorization object

```
void add_child (Authorization_Object *parent,
             Authorization_Object *child);
```

Adds the sub authorization object hierarchy rooted at the authorization object pointed to by "child" as a child of the specified parent authorization object. A particular sub authorization object hierarchy may be repeated at a number of nodes in an authorization object hierarchy.

3.1.4 Associating data files with authorization objects

```
void associate_data_file
        (Authorization_Object *an_authorization_object,
         char *file_path_name);
```

Associates an existing data file with a node in the authorization object hierarchy. A number of files may be associated with an authorization object. An authorization specified on an authorization object will apply to all the files associated with the authorization object.

```
void associate_tool
        (Authorization_Object *an_authorization_object,
         char *tool_name);
```

Associates the data files created by the specified tool with the specified authorization object.

3.1.5 Retrieving an authorization object

```
Authorization_Object *find_authorization_object
        (Authorization_Object *root=0, char *name);
```

Returns a pointer to an authorization object with the specified name, in the authorization type hierarchy pointed to by "root". If no root is specified, a pointer to a root of an authorization object hierarchy with the specified name is returned.

3.1.6 Retrieving the children of an authorization object

```
Authorization_Object_List get_children
        (Authorization_Object *an_authorization_object);
```

Returns a list of pointers to authorization objects that are children of the specified authorization object.

3.2 Mechanisms to manipulate the authorization role hierarchy

3.2.1 Creating an authorization role

```
    Authorization_Role *create_authorization_role
                        (Authorization_Role *parent=0, char *name);
```

Creates a new authorization role, as a child of the specified parent authorization role. If no parent authorization role is specified then the root node of a new authorization role hierarchy is created.

3.2.2 Deleting an authorization role

```
    void delete_authorization_role
                (Authorization_Role *an_authorization_role);
```

Deletes the subset of the authorization role hierarchy rooted at the specified authorization role from the authorization role hierarchy.

3.2.3 Adding a child to an authorization role

```
    void add_child (Authorization_Role *parent,
                Authorization_Role *child);
```

Adds the sub authorization role hierarchy rooted at the authorization role pointed to by "child" as a child of the specified parent authorization role. A particular sub authorization role hierarchy may be repeated at a number of nodes in the authorization role hierarchy.

3.2.4 Associating users with authorization roles

```
    void associate_user
                (Authorization_Role *an_authorization_role,
                 char *user_name);
```

Associates a user with an authorization role. A number of users may be associated with an authorization role. An authorization specified on an authorization role will apply to all the users associated with the authorization role.

3.2.5 Retrieving an authorization role

```
    Authorization_Role *find_authorization_role
              (Authorization_Role *root=0, char *name);
```

Returns a pointer to an authorization role with the specified name, in the authorization type hierarchy pointed to by "root". If no root is specified, a pointer to a root of an authorization role hierarchy with the specified name is returned.

3.2.6 Retrieving the children of an authorization role

```
    Authorization_Role_List get_children
              (Authorization_Role *an_authorization_role);
```

Returns a list of pointers to authorization roles that are children of the specified authorization role.

3.3 Mechanisms to manipulate the authorization type hierarchy

3.3.1 Retrieving an authorization type

```
    Authorization_Type *find_authorization_type
              (Authorization_Type *root=0, char *name);
```

Returns a pointer to an authorization type with the specified name, in the authorization type hierarchy pointed to by "root". If no root is specified, a pointer to a root of an authorization type hierarchy with the specified name is returned.

3.3.2 Retrieving the children of a node in the authorization type hierarchy

```
    Authorization_Type_List get_children
                (Authorization_Type *an_authorization_type);
```

Returns a list of pointers to authorization types that are children of the specified authorization type.

3.4 Mechanisms to grant and revoke authorizations

3.4.1 Granting authorizations

```
    int grant_authorization
                (Authorization_Object *an_authorization_object,
                 Authorization_Role *an_authorization_role,
                 Authorization_Type *an_authorization_type);
```

Grants an authorization of type "an_authorization_type" on the object pointed to by "an_authorization_object" to the authorization role pointed to by "an_authorization_role".

3.4.2 Revoking authorizations

```
    int revoke_authorization
                (Authorization_Object *an_authorization_object,
                 Authorization_Role *an_authorization_role,
                 Authorization_Type *an_authorization_type);
```

Revokes an authorization of type "an_authorization_type" on the object pointed to by "an_authorization_object" from the authorization role pointed to by "an_authorization_role".

4 Implementation Strategy

The implementation of the authorization 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 authorization model will be implemented in one representative tool in each functional area of the RASSP design process, to demonstrate the value of the common authorization model. The authorization information captured in these systems will be modeled using the STEP AP203 [ISO, 1993] configuration management (CM) conformance class. This would allow the authorization and CM information associated with a product to be exchanged seamlessly between the various tools used during the life cycle of the product. Rockwell Inc. will evaluate the STEP AP203-- CM conformance class for its adequacy in modeling the authorization information captured by the enterprise framework tools. They will also propose extensions to the AP203--CM conformance class where needed, and work with PDES Inc.[3] to incorporate the extensions into the standard.

4.1 Schedule

Build 1

Detailed implementation plan for enterprise framework tools

(Enterprise framework tool vendors) : Dec. 31, 1994

Detailed implementation plan for CAD tools (CAD tool vendors) : Dec. 31, 1994

Build 2

Implementation of the Authorization Model

(Enterprise framework tool vendors, CAD tool vendors) : Dec. 31, 1995

Build 3

Test and validate the Authorization Model implementation

(Martin Marietta) : June 30, 1996

[3]PDES is the Product Data Exchange Using STEP, an industrial consortium that develops the STEP standards.

Build 4

Implement improvements to the Authorization Model

(Enterprise framework tool vendors, CAD tool vendors) : Dec. 31, 1996

References

[Intergraph, 1993] Intergraph, *Intergraph/Network File Manager -- Administrator's Reference Manual*, Huntsville, Alabama, 1993.

[ISO, 1993] International Standards Organization, *Configuration Controlled 3D Designs of Mechanical Parts and Assemblies*, ISO 10303-203, Fairfax, Virginia: U.S. Product Data Association, 1993.

[Martin Marietta, 1994] Martin Marietta Laboratories, "The Configuration Management Model for the RASSP System," Moorestown, New Jersey, September, 1994.

[Rabitti, 1991] Rabitti, F., Bertino, E., Kim, W., and Woelk, D., "A Model of Authorization for Next-Generation Database Systems," *ACM Transactions on Database Systems* 16(1): 88-131, March, 1991.