

The PCA Metadata System

Georgia Institute of Technology
and
Space and Naval Warfare Systems Center San Diego

Version 1.0
March 2, 2004



©2004 Georgia Tech Research Corporation, all rights reserved.

A non-exclusive, non-royalty bearing license is hereby granted to all persons to copy, modify, distribute and produce derivative works for any purpose, provided that this copyright notice and following disclaimer appear on all copies: THIS LICENSE INCLUDES NO WARRANTIES, EXPRESSED OR IMPLIED, WHETHER ORAL OR WRITTEN, WITH RESPECT TO THE SOFTWARE OR OTHER MATERIAL INCLUDING, BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE, OR ARISING FROM A COURSE OF PERFORMANCE OR DEALING, OR FROM USAGE OR TRADE, OR OF NON-INFRINGEMENT OF ANY PATENTS OF THIRD PARTIES. THE INFORMATION IN THIS DOCUMENT SHOULD NOT BE CONSTRUED AS A COMMITMENT OF DEVELOPMENT BY ANY OF THE ABOVE PARTIES.

This material is based in part upon work supported by the U.S. Defense Advanced Research Projects Agency (DARPA) and other agencies of the U.S. Department of Defense (DoD). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA or the DoD.

The US Government has a license under these copyrights, and this material may be reproduced by or for the U.S. Government.



Acknowledgements

Authors

The primary authors of this document were:

Daniel P. Campbell	Georgia Tech Research Institute
Dennis M. Cattel	Space and Naval Warfare Systems Center San Diego
Randall R. Judd	Space and Naval Warfare Systems Center San Diego
Mark A. Richards	Georgia Institute of Technology

The authors wish to thank the members of the Morphware Forum who reviewed and contributed to this document. The authors also thank the Defense Advanced Research Projects Agency (DARPA) and the US Navy's Space and Naval Warfare Systems Center San Diego for their support of this work.

Document Change History

This is the initial public release.

Table Of Contents

Acknowledgements.....	i
Authors.....	i
Document Change History.....	ii
Table Of Contents.....	iii
1. Introduction.....	1
1.1. Background.....	1
1.2. Approach.....	2
2. Requirements.....	3
3. The PCA Metadata System.....	5
3.1. Metadata Content Description.....	5
3.2. Metadata XML Representation.....	7
4. An XML Example.....	9
5. References.....	11
6. Appendix: Additional Metadata System Issues.....	12

1. Introduction

The Polymorphous Computing Architectures (PCA) program [1] is a U.S. Defense Advanced Research Projects Agency (DARPA) effort to develop integrated computing systems with architectures that can be changed, or *morphed*, to closely match the resource requirements of Department of Defense (DoD) application programs. Those resources can include computation, memory, and communication capabilities.

The PCA program also explicitly addresses the issue of software commonality between the various PCA hardware systems so that DoD applications can be moved from one system to another with minimal software modifications and minimal impact on performance. To carry out this function, the PCA program has created the Morphware Forum [2], a program-wide activity meeting quarterly to debate and develop software concepts and standards.

This document describes the standard approach to handling PCA-related metadata as defined by the Morphware Forum. Specific metadata content is defined in other documents establishing the Morphware Stable Interface (MSI) and standardized by the Forum.

1.1. Background

Within the PCA program, the term *metadata* refers to any information needed to create, build, and run a PCA application, that is not contained in the source code or the executable binary. This broad definition of metadata includes, for instance, such information as file dependencies and compiler options normally kept in UNIX Makefiles. Some metadata, such as that in the Makefiles, continues to be recorded in special formats as determined by other development tools. However, in those cases where the metadata is unique to the PCA program, the metadata system described in this document was designed to record the metadata and to transfer metadata information from one program or tool to another.

Various kinds of PCA information are included in the broad term metadata. They include simple data, such as program parameters, as well as complex data defining the arbitrary graphs that are used to describe PCA architectures and interconnections among programs in a large application. The information in the metadata system can also be dynamic: programs can get run-time configuration information, modify metadata at run time, and access real-time status variables such as the current operating temperature.

PCA metadata may be accessed or modified throughout the development and execution of a PCA application. For example, PCA compilers manipulate metadata at compile time, the PCA run-time system uses and creates metadata at application load time, and the application itself will access metadata at run time. In addition, various tools manipulate PCA metadata throughout the process of development and deployment of an application.

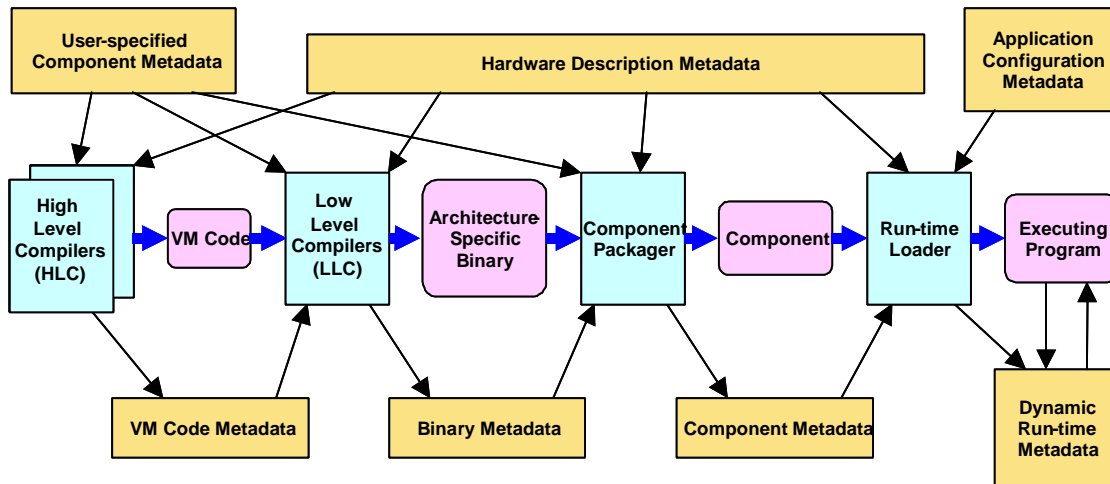


Figure 1. A possible PCA development environment illustrating that there are a number of different metadata groupings (contexts), and that each program concerns itself with more than one context.

The illustration in Figure 1 shows a typical example of how the various aspects of a PCA development system might work together. There are a number of different groupings of metadata used for different purposes. Each of the programs that use metadata works with more than one of these groupings. It is clear that throughout the PCA program there must be a single metadata system for manipulating metadata so that users, application writers, and tool developers do not have to work with multiple metadata formats.

1.2. Approach

Each of the metadata groupings described in the paragraph above and in Figure 1 is called a *metadata context*. The purpose of the metadata system is to provide a defined way to document and handle metadata for each of the contexts used in the PCA program.

The PCA approach to handling metadata information is to separate the problem into two orthogonal tasks:

- Design a simple, flexible, extensible, and usable common metadata system.
- For each metadata context, define what metadata is required and map it onto the common metadata system.

This document describes the single standard metadata system used by the PCA program. Other documents standardized by the Morphware Forum describe the actual metadata content for specific contexts.

2. Requirements

The metadata system must meet the following general requirements:

- R1. The system must allow metadata to be accessed or modified during compile and link time, at application load time, during execution of the application, and at any other time during program development.
- R2. The system must be extensible so that new kinds of metadata can be added without breaking existing programs and applications.
- R3. The system must allow programs to create and modify metadata as well as to access it.
- R4. The system must be independent of programming language, hardware architecture, and internal implementation.

The following additional requirements must be met when the system is used by an executing PCA application:

- R5. Programs must access *current* metadata values, for instance, after a morph or when the metadata value represents current performance metrics or environment measurements.
- R6. Programs must be able to request notification when a metadata value changes.
- R7. The system must provide acceptable performance in terms of speed of operation and memory resources used.

During discussions among PCA participants, a number of other capabilities were brought up that were not included in the final requirements. Among them:

- The system does not provide a capability for configurability of the metadata. Such a feature could allow expressions of metadata values (*e.g.*, “`$some_value = 2 * $other_value`”) and conditional statements (*e.g.*, “`if $display then load GUI`”). These capabilities are handled with other tools.
- Access controls for reading and writing the metadata are not included in the system. The design and implementation of such a facility would require resources beyond that available to the PCA program, and would add considerable complexity to the system.
- The system does not attempt to specify a general approach to querying the contents of the metadata “database.” For specific cases, features such as this can be implemented using the capabilities provided by the system.
- Large binary objects were considered for inclusion directly as metadata. In the final system, large objects are stored separately, and normal metadata text values provide the locating information (*e.g.*, file pathnames).
- An inherent capability for describing ranges of values or constraints was suggested, but in the final system, this information is provided with additional metadata items when needed. For instance, two metadata entries could be used to contain the minimum and maximum values.

REQUIREMENTS

- The system does not validate the metadata content. When wanted, this capability is provided with other tools.

3. The PCA Metadata System

The PCA metadata system provides a standard way to document and store the various metadata contexts that will be used throughout the PCA program. This system consists of two parts. The first part is documentation that describes all metadata content in terms of *objects*, *parameters* of those objects, and *references* to other objects. The second part is a standard mapping for these metadata constructs into Extensible Markup Language (XML) [3]. XML is a flexible text format for data and document interchange standardized by the World Wide Web Consortium (W3C) [4].

The PCA metadata system does not specify the Application Program Interface (API) used by programs to manipulate metadata. Rather, the format of how metadata objects are stored in XML is standardized. This allows developers and users to take advantage of the many free and commercial tools for XML *data binding* [5] to build APIs that are specific to their requirements and implementation language. These tools, given a description of the context, can automatically generate a language-specific API to access and modify the XML content. For instance, a developer writing a compiler in Java could choose to use Java Architecture for XML Binding (JAXB) [6] tools to automatically generate a set of Java classes that directly map to the compiler's metadata. The developer of each tool needing to read or write metadata in XML format will need to acquire a free or commercial tool to generate an API for the language they are using, or they could choose to implement a custom API themselves.

Ultimately, the complete and formal specification of the allowed content for a particular metadata context will be given by an XML *schema*. XML schemas provide a means for defining the structure and content of XML documents. The XML Schema language is a W3C standard [7].

3.1. Metadata Content Description

All metadata content is described in terms of objects and parameters of those objects. Parameters can be various scalar types, other objects, references to other objects, and ordered lists of parameters. This system is capable of describing metadata of arbitrary complexity including graphs of objects. For instance, object references may be used to describe a parent-child hierarchy among the objects in the metadata. Also note that an object may be entirely contained in another object allowing hierarchical design decisions to be reflected in the metadata content if desired.

For each metadata context, the allowed metadata content is documented using whatever format is appropriate for the data, usually a combination of diagrams, tables, and text.

For example, suppose the metadata in an example metadata context must describe an architecture family that has one or more processors and one or more memories connected to a single bus as illustrated in Figure 2.

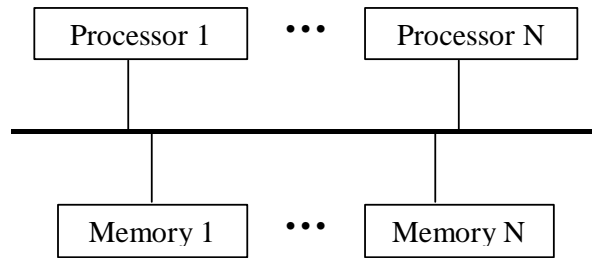


Figure 2. A family of computer architectures with one or more processors and one or more memories connected to a single bus.

One possible way to document the metadata in this context is shown in Figure 3. Parameters have been added to each object for illustration purposes.

Note: The context documentation example in Figure 3 uses a Unified Modeling Language (UML) diagram [8]. UML is a graphical modeling language standardized by the Object Management Group (OMG) [9]. The use of UML diagrams for metadata content documentation is not a requirement of the metadata system. In Figure 3, metadata objects are shown as classes with attributes but no methods. Object parameters are shown using class attributes. Parameters that are references to other objects are represented with associations between classes. Parameter names for object references are given using the UML role name for the association. Lists of object references are used when the association’s multiplicity designates the possibility of more than one reference, and all such lists are ordered. An object contained in another object is shown with a composition relation.

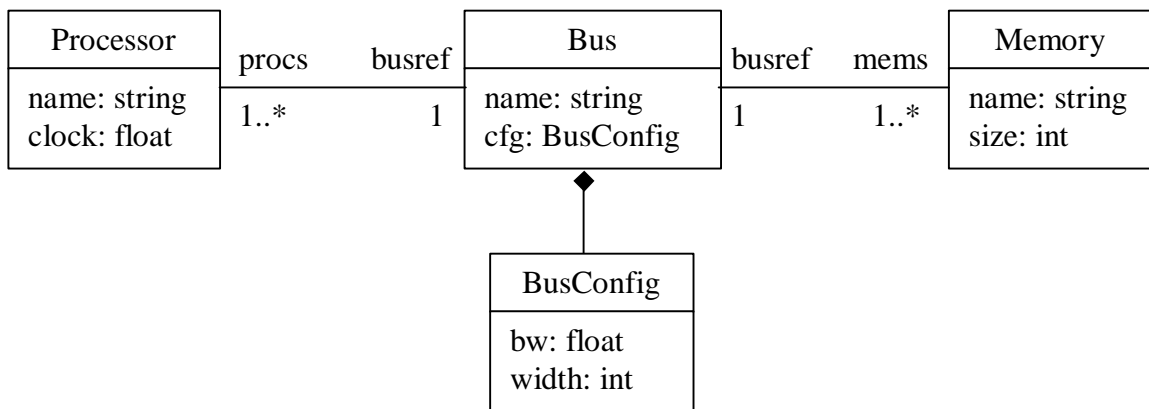


Figure 3. A diagram documenting the metadata context for describing the family of systems shown in Figure 2.

Figure 3 shows that metadata in this context is allowed to consist of objects of four types. Processor objects have three parameters: a string name; clock, of type float; and busref, a reference to a Bus object. Memory objects have three parameters: a string

name; size, of type `int`; and `busref`, a reference to a Bus object. Bus objects have four parameters: a string `name`; `procs`, a list of one or more references to Processor objects; `mems`, a list of one or more references to Memory objects; and `cfg`, of type `BusConfig` where a `BusConfig` is an object with two parameters of its own: `bw`, of type `float`, and `width`, of type `int`.

Objects do not necessarily have references to other objects. For instance, an object may always be contained in other objects, as is the `BusConfig` object in Figure 3. Or, one or more objects might be used solely to contain parameters describing global metadata values.

3.2. Metadata XML Representation

As pointed out above, the PCA program requires the transfer of metadata from one application to another. To facilitate this, as well as to minimize development time and the use of program resources, the transfer format must be standardized. There are a number of reasons why this format should be text; among them, it allows the metadata to be created and modified with standard text manipulation tools, and it provides a human-readable document for inspection and problem solving. Fortunately, there exists a widely used, standard text format, XML, standardized by the W3C.

PCA metadata will be represented in an XML format that maps PCA context documentation to XML in a standard way. This allows a user to easily predict the XML form from the documentation, or to derive the metadata context from the XML. The choice of XML also allows the use of many available tools to manipulate PCA metadata.

XML requires that there be exactly one top-level element, defined here as `<pca_metadata>`. An additional sub-element will be used to identify PCA metadata contexts (for example, `<example_context>`). Top-level metadata objects appear as elements within this context element where the object element start tag corresponds to the object type name in the context documentation. Each object element has an attribute called `ident` with a string value that uniquely identifies the object and that may be used to reference this object. This attribute is associated with the `ID/IDREF` feature of the XML Document Type Description (DTD) format and the XML Schema specification.

Within an object element, each object parameter is described using an element with a start tag that is the same as the parameter name. Scalar parameter values are given as text in the content of the element, and may be strings, integers, and floating point numbers. Object references are empty elements that have an attribute `objref` with a string value the same as that of the `ident` attribute of the object being referenced. A parameter that is another object (not an object reference) is represented as an element nested within its containing object element. A parameter may appear more than once within an object to designate a variable list of values. The order of elements in a list is maintained when reading and writing the metadata.

Figure 4 summarizes at a high level how metadata information corresponds to the elements of an XML document.

THE PCA METADATA SYSTEM

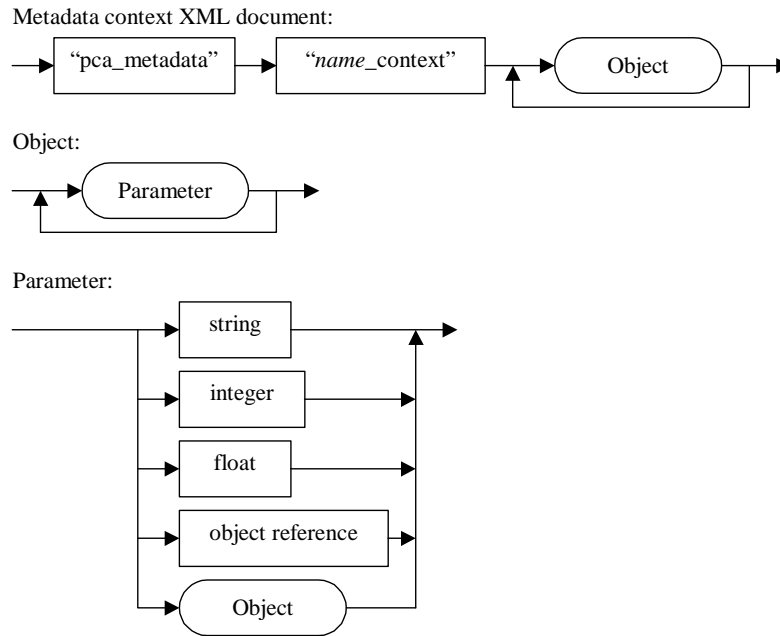


Figure 4. A high-level depiction of how an XML document is used to represent a metadata context. Each box is an XML element. Rectangular boxes are terminals (they have no further description) while rounded boxes have an additional sub-diagram describing their contents.

4. An XML Example

Suppose it is necessary to describe a specific system, from the family of systems shown in Figure 1 and Figure 3, that has exactly two processors and two memories. The XML file containing the metadata for this system is shown in Figure 5.

```

<?xml version="1.0"?>
<pca_metadata>
  <example_context>
    <Processor ident="proc001" >
      <name> proc1 </name>
      <busref objref="bus001" />
      <clock> 1.5e9 </clock>
    </Processor>
    <Processor ident="proc002" >
      <name> proc2 </name>
      <busref objref="bus001" />
      <clock> 2.0e9 </clock>
    </Processor>
    <Bus ident="bus001" >
      <name> bus </name>
      <procs objref="proc001" />
      <procs objref="proc002" />
      <mems objref="mem001" />
      <mems objref="mem002" />
      <cfg>
        <BusConfig ident="busconfig001" >
          <bw> 1.0e9 </bw>
          <width> 128 </width>
        </BusConfig>
      </cfg>
    </Bus>
    <Memory ident="mem001" >
      <name> mem1 </name>
      <busref objref="bus001" />
      <size> 512 </size>
    </Memory >
    <Memory ident="mem002" >
      <name> mem2 </name>
      <busref objref="bus001" />
      <size > 512 </size>
    </Memory >
  </example_context>
</pca_metadata>

```

Figure 5. XML description of the metadata for a two-processor, two-memory system. A diagram describing the metadata context for this family of systems is shown in Figure 3.

If it is ever necessary to commingle PCA metadata files with XML files from other sources, an XML *namespace* may be used to ensure that there are no name conflicts. This is done by adding a “pca” prefix and colon to all PCA elements and attributes, and then identifying the namespace prefix by adding an attribute to the `pca_metadata` element. The beginning of the XML metadata file in Figure 5 would then be:

AN XML EXAMPLE

```
<pca:pca_metadata xmlns:pca="http://www.darpa.mil/ipto/pca/">  
  <pca:example_context>  
    <pca:Processor pca:ident="proc001">  
  ...
```

5. References

1. The Polymorphous Computing Architectures (PCA) Program, Information Processing Technology Office (IPTO), Defense Advanced Research Projects Agency (DARPA), www.darpa.mil/ipto/Programs/pca/index.htm.
2. The PCA Morphware Forum, www.morphware.org.
3. The Extensible Markup Language (XML), World Wide Web Consortium, www.w3.org/XML.
4. The World Wide Web Consortium (W3C), www.w3.org.
5. Ronald Bourret, "XML Data Binding Resources," October 6, 2003. Available at www.rpbouret.com/xml/XMLDataBinding.htm.
6. Java Architecture for XML Binding (JAXB), java.sun.com/xml/jaxb.
7. XML Schema, World Wide Web Consortium, www.w3.org/XML/Schema.
8. The Unified Modeling Language (UML), Object Management Group, www.omg.org/technology/documents/formal/uml.htm.
9. The Object Management Group (OMG), www.omg.org.

6. Appendix: Additional Metadata System Issues

This section discusses additional issues related to the metadata system. These are outside the scope of this metadata system document but are recorded here for reference.

Procedural interface. Some PCA participants expressed concern that the metadata system should be procedural rather than declarative. The thought was that this would give the compiler direct access to metadata values allowing for the possibility of some sophisticated optimizations. It appears, however, that this capability can be layered above the current declarative approach. For instance, a compiler could recognize the API calls used by its source language to access metadata, and then read the metadata values directly. Further, this capability would be implementation specific and so it is not suitable for standardization by the PCA program. Consequently, this standard metadata system does not further address the procedural concepts.

Change notification (R6). The requirement that a running PCA application be able to request notification of a change in the value of a metadata item is not directly addressed by this standard as this document does not standardize the run-time API. Nevertheless, it is expected that this capability will be provided by the various run-time interfaces provided by PCA vendors. The notification requirement may mean that tool developers will have to extend an automatically generated API, as some XML data-binding tools may not include this feature.

Version compatibility (R2). In practical use, programs that have previously been developed for an earlier definition of a metadata context will have to access metadata from a newer, updated context. If the metadata format were too rigid, this scenario would cause the program to fail, an unacceptable result. The PCA XML format described here is general enough to allow new types of metadata to be added without causing the XML parsers to fail. Many of the data binding tools will validate that the XML conforms to the schema of a particular context, but they also allow this validation to be skipped in cases where it is undesirable.

Specification time. Some metadata values may be specified early in development, and others may not be known until, say, application execution time. The runtime system must provide a non-fatal way for a program to determine whether a metadata value exists. If additional information is available to the program designer that tells when in the process a metadata value will be specified, that information is assigned to additional metadata items on a case-by-case basis.

Persistence of changes. There is no underlying database that preserves metadata values across program invocations. If a program changes a metadata value, the change does not persist after that program terminates unless the metadata context is explicitly written to a file.

Scope. A program may consist of multiple threads, possibly running on multiple processors. It is possible that a metadata item could have a different value when read by different threads (*e.g.*, processor clock speed, thread role). A change to a metadata value

made by one thread need not be propagated to other threads and processes in the application.

Read-only values. Certain values in a metadata context may be defined by the context documentation to be read only. However, there need not be any runtime support for this concept. It is up to the application writer to manipulate metadata within the semantics specified by the context documentation. The result of attempting to write a read-only value is implementation dependent.