From the Proc. of the 1998 Inter. Conf. on Software Maintenance, Bethesda, Maryland, USA, November 1998

Evaluating Software Deployment Languages and Schema

An Experience Report

Richard S. Hall, Dennis M. Heimbigner, Alexander L. Wolf
Department of Computer Science
University of Colorado
Boulder, CO 80309 USA
{rickhall,dennis,alw}@cs.colorado.edu

Abstract

Software distribution is evolving from a physical media approach to one where it is practical and advantageous to leverage the connectivity of networks. Network distribution of software systems provides timeliness and continuity of evolution not possible with physical media distribution methods. To support network-based software distribution, companies and organizations such as Microsoft, Marimba, and the Desktop Management Task Force (DMTF) are strengthening their efforts to package software systems in a way that is conducive to network distribution and management. The result of these efforts has led to the creation of software description languages and schema such as the Open Software Description format created by Microsoft and Marimba and the Management Information Format created by DMTF. While these efforts are steps in the right direction, they do not address deployment issues in a complete and systematic fashion. The contribution of this paper is to evaluate these leading software description technologies.

1. Introduction

The connectivity of large networks, such as the Internet, is affecting how software deployment is being performed. The simple notion of providing a complete installation procedure for a software system on a CD-ROM is giving way to a more sophisticated notion where software deployment involves the ongoing cooperation and negotiation among software producers themselves and among software producers and software consumers. This new cooperation has led to exciting possibilities in software deployment that were not previously possible.

One result of this connectivity and cooperation is the possibility of software producers offering high-level deployment services to their customers. The responsibility of the software producer is now moving beyond the mere installation of their software systems to encompass other activities in the software deployment life cycle. The resulting benefit to the software consumer is a lowered total cost of ownership because less effort need be expended on maintaining the software that they own.

One way these new notions of connectivity and cooperation are being addressed is through software system and component languages and corresponding schema. The purpose of these languages and schema is to provide enough semantic information about a given software system so that many standard deployment tasks can be automated or simplified. Specific efforts such as the Software Dock [4], the Open Software Description (OSD) [8], and the Desktop Management Task Force (DMTF) [2] are trying to create a standard syntax and semantic for describing software systems in order to facilitate deployment and related activities.

The purpose of this paper is to discuss two specific high-profile deployment language efforts. One is Microsoft's and Marimba's joint proposal named OSD [8] that was submitted to the W3 Consortium. The other is the DTMF consortium's Software MIF standard [2]. In order to discuss these technologies, the next section briefly introduces the requirements and issues of software deployment. Sections 3 and 4 discuss OSD and MIF, respectively, and specifically address their solutions with respect to the requirements and issues outlined in Section 2.

2. Requirements for software deployment languages and schema

Software deployment is not a simple, single process to be performed after a software system has been developed. Software deployment is a collection of interrelated activities that address all of the issues of interfacing a deployed software system to the ongoing usage of the consumer as well as the ongoing development efforts of the producer. These

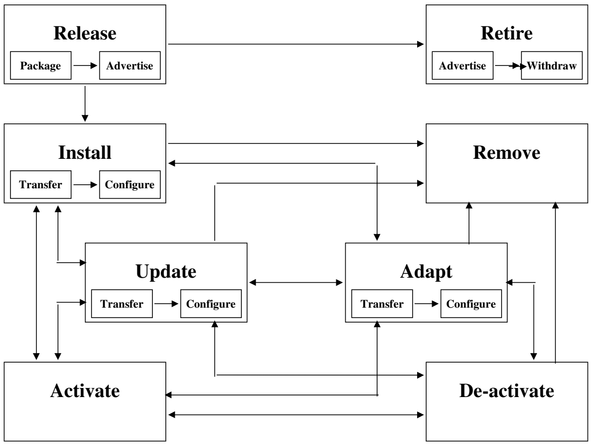

Figure 1. Software Deployment Life Cycle

activities are collectively referred to as the software deployment life cycle and their relationships are depicted in Figure 1. For a more detailed description of the software deployment life cycle see [5]. A brief description of each life cycle activity follows:

- **Release** - package, advertise, and make a software system ready for release.
- **Install** - gather and configure the necessary artifacts of a software system in order for it to be used on a consumer site.
- **Update** - reconfigure a software system at a consumer site in response to producer-side changes.
- **Adapt** - reconfigure a software system at a consumer site in response to consumer-side changes.
- **Activate** - transform the software system into a usable state (i.e., execute it).
- **De-activate** - return the software system to a dormant state (i.e., shut it down).
- **Remove** - remove a given software system from a consumer site.
- **Retire** - retire a given software system, thus making it unavailable for future deployment.

These activities define the scope of the software deployment problem space; a space for which software deployment languages and schema must provide coverage. Taking into account the above description of the software deployment life cycle, certain requirements for a software deployment language or schema emerge. Specifically, a software deployment language or schema must be able to:

- semantically describe software systems and components and
- semantically describe consumer sites where software is to be deployed.

The purpose of these descriptions is to provide semantic knowledge that is rich and rigorous enough to support the automation of the software deployment life cycle. Given such descriptions, generic solutions to software deployment tasks are possible by combining software product knowledge with consumer site knowledge. For example, a generic software installation process interprets the constraints, dependencies, and various configurations of a specific software system with respect to the constraints and resources available at a particular consumer site; through this interpretation the generic software installation process performs a specific installation activity for a given software system.

In order to provide such semantic descriptions of software systems and computing sites it is necessary to adopt a semantic model. A common approach, and one that is assumed in this paper, is that software systems and computing sites are objects that can be modeled as a collection of attributes or properties. These collections of attributes may have internal structure, but at the base level they map to primitive attribute types (e.g., integer, string). This model is used throughout this paper and, in general, is used by both OSD and MIF.

2.1. Consumer site description

There are two classes of participants in the software deployment problem space, namely producers and consumers. The purpose of the consumer description is to provide a context about the site into which a software system is to be situated; this is essential to fully describe a software system or component. For example, it is difficult to describe a software system's dependency on properties such as operating system, architecture, or dependent software subsystems if there is no accessible model of such information. As such, the consumer site description and the software system description, should be viewed as two halves to a whole, rather than two distinct entities.

At the very least, the consumer site description must record information about the state of the consumer site at any given time; such records include the following:

- **Properties** - attributes of the consumer site that may affect the outcome of deployment operations (e.g., operating system, hardware architecture, and hardware configuration such as memory),
- **Constraints** - restrictions placed on the values of specific site properties either by deployed software systems or by administrators (e.g., a deployed software

system may constrain how the operating system can be configured or which version of a shared resource can be installed), and

- **Resources** - interfaces and capabilities provided by the consumer site as well as the existence and availability of other deployed software systems at the site.

2.2. Software system and component description

The other half of the software deployment problem space is represented by the software producers' descriptions of software systems and components. These descriptions define an interface to the products and services provided by software producers. The goal for the software system description is to provide enough semantic information about a particular software system so that it can be deployed in an automated fashion. The responsibility of describing a software system lies solely with the producer of the software system since the producer has the most knowledge about the system.

We have identified minimally [6, 5] five classes of semantic information that must be described for a software system or component, these are:

- **Assert constraints:** These are predicates over required attribute values where the predicate must evaluate to true or else an unresolvable conflict is indicated. Examples include the requirement of Windows 95 or that total memory is greater than 24 megabytes. In general, assertion constraints are used for two purposes, selecting a properly configured software system and to maintain the proper operation of a deployed software system; the former case being the most common. The latter case requires the notion of a consumer site model where assert constraints can be maintained in order to enforce and disallow incompatible changes to the consumer site.

- **Dependency constraints:** These are also predicates over required attribute values. But in contrast to assert constraints, each dependency constraint has an associated resolution procedure that can be invoked to attempt to resolve conflicts in the case that the dependency predicate evaluates to false. The difference between assert constraints and dependency constraints is subjective because although a constraint may be solvable it may not be practical to solve it in all cases. Thus a particular constraint may be an assert in some cases and a dependency in others. Dependency constraints may be used to express dependencies on subsystems that can installed if not available, on properties that can be reconfigured if the values are not appropriate, or on generic capabilities required by the software system.

- **Artifacts:** These are the physical components, represented as files, that comprise a software system. Artifacts can be executables, libraries, data sets, or documentation. It is necessary to understand the source, destination, description, type, and mutability of the artifacts that comprise a software system.

- **Configuration:** This maps sets of properties to associated sets of artifacts that must be present depending on the value of the properties. This mapping is combined with the assert and dependency constraints to select valid configurations for a given software system. A configuration description for a software system may include functional properties like optional subcomponents or non-functional properties such as performance characteristics.

- **Activities:** These are the specialized activities that are required in order to complete deployment. It is unlikely that a standard deployment language or schema can address everything that is ever going to be needed to perform software deployment tasks, therefore some notion of describing and including specialized tasks is necessary.

3. Open software description

The Open Software Description (OSD) is a W3 Consortium proposed standard created jointly by Microsoft and Marimba. In general, OSD provides a vocabulary for packaging software; this includes describing software components, their versions, their underlying structure, and relationships among components. OSD, which is one piece of Microsoft's Zero Administration Initiative [7], is related to Microsoft's Channel Definition Format (CDF)[3] for "push" content. The two standards combined are intended to be used in conjunction with "push" technologies to enable software systems to automatically install and update themselves. The syntax for both OSD and CDF are based on the Extensible Markup Language (XML) [1]. Refer to Appendix A for a simple OSD specification.

3.1. Detailed description

The OSD syntax is organized hierarchically with very few keywords. The major syntax elements for OSD include: *SOFTPKG* to define a general software package, *IMPLEMENTATION* to describe an implementation of a software package, and *DEPENDENCY* to indicate a dependency on some other software package or package component.

A *SOFTPKG* has *IMPLEMENTATION* elements as its children to describe specific implementations of the software package. A *SOFTPKG* may also have *DEPENDENCY* elements as its children to specify required software packages. Additional children of *SOFTPKG* include: *TITLE* to indicate the "friendly name" of the software package, *ABSTRACT* to provide a short description of the nature and purpose of the software package, and *LICENSE* to indicate the location of the license agreement or copyright noticed to be retrieved.

An *IMPLEMENTATION* element may also have *DEPENDENCY* elements as children in order to specify dependent components. Additional children of *IMPLEMENTATION* include: *CODEBASE* to indicate the location of the archive for the implementation, OS to indicate the required operating system, *PROCESSOR* to indicate the required central processing unit, *LANGUAGE* to indicate the required natural language in the software's user interface, *VM* to indicate the required virtual machine, *MEMSIZE* to indicate the required amount of run-time memory, DISKSIZE to indicate the required amount of disk space, and *IMPLTYPE* to indicate the type of the implementation. The *OS* element may also have an *OSVERSION* as a child to indicate the required version of the required operating system.

Wherever used, the *DEPENDENCY* element can be either an assert or install dependency. For assert, if the dependency does not exist, the entire dependency fails. For install, if the dependency does not exist an attempt will be made to install it.

3.2. Evaluation

In all fairness, it must be stated up front that OSD is an initial attempt by Microsoft and Marimba to create a joint standard for describing software systems and components for deployment. The intention of OSD is not expressly to address the entire software deployment life cycle described within this paper, but more specifically to address the installation and update processes using "push" technologies. In that regard it is certainly a step in the right direction. Despite this modest goal, though, it is clear than many of the same requirements outlined in Section 2 need to be addressed by OSD at some point to make it useful for a larger class of problems. As it stands, OSD currently addresses only a limited form of installation and update in the software deployment life cycle.

Inspecting the OSD specification [8], it is clear that there are two halves to the specification: the Microsoft half and the Marimba half. Given the issues surrounding the collaboration and the time constraints involved in dealing with market forces, it appears that this initial attempt was merely Microsoft and Marimba combining their separate simple, description efforts into one standard. In particular, the Microsoft half is concerned with some consumer side properties such as operating system type and version, while the Marimba side is simply considered a different implementation type from the Windows implementation type assumed by Microsoft.

The main thrust of OSD is to describe a single version of a software system. Standardized handling of a handful of consumer side properties and of dependent systems and components makes this possible. A single OSD description may contain any number of variants or implementations based on the properties provided. In this scenario Java is simply treated as a different implementation. In addition, the actual processing to be performed on an OSD specification is completely unspecified within the description, nor is there enough detail to completely infer the processing. Some of the OSD specifics are detailed below.

Consumer site description: OSD lacks an explicit consumer site model, thus making it difficult to use assert constraints to maintain a properly working configuration of a software system. At the most basic level, OSD does not provide any standardized mechanism for answering questions as simple as which operating system is running on the consumer site or which microprocessor architecture is in use, although it assumes that these answers are required. Since there is no standardized means to answer these simple questions, there is no hope of asking more complicated questions. Questions such as which software will break if a specific property is changed, are far beyond the capabilities of OSD. It is assumed that Microsoft intends to use the Registry component of Windows to answer basic questions for Windows-centric platforms and that Marimba is not generally concerned with these types of issues since they are concentrating on a platform neutral environment (i.e., Java). Neither approach is particularly well-suited for the task at hand because the Windows Registry has no inherent schema definition for deployment and Java is not as platform neutral as one might hope.

Assert constraints: The OSD specification directly supports only a limited set of assert constraints. The specification indicates that OSD can be arbitrarily extended by third parties using separate XML namespaces, but this approach will result in a fragmented solution to software deployment if each interested producer produces a proprietary schema. The schema must be enriched and broadened to avoid the same situation that currently exists with many software deployment tools using proprietary schema. In such a scenario, the only value that OSD provides is combining all of the schema into one common syntax, which is a tenuous contribution at best.

The current set of assert constraints in OSD appear to have limited usefulness. Their intended use appears to be to select an implementation and nothing more. In other words, the supported assert constraints are generally static consumer side properties, that will only ever be checked once when installing or updating and then forgotten. Further, there is no support for using constraints to maintain the integrity or operational correctness of a software system. It is particularly vague, for example, how changing the operating system version would affect already deployed software systems.

Dependency constraints: OSD does support the notion of a software system and component dependencies, but these are merely URL pointers to another OSD specification. The semantics of a dependency in OSD are particularly weak and the dependency discussion of Section 4.2 is applicable here too.

Artifacts, Configuration, and Activities: Artifact description is absent from the OSD specification. In general it is always assumed that an OSD specification merely points to an archive to be installed. Nor is there any support in OSD for configurable software systems. The software system being described is either considered monolithic with respect to its archive or that any configuration occurs in an external installation or update process. Lastly, the support provided by OSD is clearly not sufficient to perform the variety of software deployment activities that are required. For all of these shortcomings it appears that the solution is to fall back to the extensibility of XML namespaces to allow each producer to introduce their own notion of process description. As before, this solution is insufficient if the true intent is to provide some form of a unified software deployment framework.

4. Management information format

The Desktop Management Task Force (DMTF) is an industry consortium chartered with the development, support, and maintenance of management standards for personal computer systems and products. An initial result of the DMTF effort was the Desktop Management Interface (DMI), which creates a common interface layer to access management information on computing systems. An effort related to DMI is the Management Information Format (MIF), which is a common, hierarchical data model used in describing all aspects of computing systems, including software systems. A major contribution by DMTF is the formation of working groups to create standard MIF schema to describe various aspects of computing systems. The focus of the discussion in this section is on the Software MIF created by the Application Management working group. Currently, DMTF is moving towards a new, object-oriented data model called the Common Information Model (CIM). The Software MIF is currently being mapped to this model. Appendix B shows a simple Software MIF specification. It is intended to be analogous to the OSD example of Appendix A. Note that the MIF specification is substantially larger than the OSD specification. This is partly because the MIF is more detailed, but also because it is more verbose as well.

4.1. Detailed description

The standard syntax of MIF is broken into three hierarchical elements: *COMPONENT*, *GROUP*, and *ATTRIBUTE*. A *COMPONENT* is a top-level grouping mechanism and is used to describe a single entity. A *COMPONENT* may contain one or more *GROUP* elements which are used to group a related set of *ATTRIBUTE*s. An *ATTRIBUTE* is a child of *GROUP* and encapsulates a typed value. The Software MIF creates a set of standard software groups to describe a software component, these *GROUP*s are named *ComponentID*, *Software Component Information*, *Software Signature*, *Location*, *Equivalence*, *Superseded Products*, *Maintenance*, *Verification*, *Subcomponents*, *Component Dependencies*, *Attribute Dependencies*, *File List*, *Installation*, *Installation Log Files*, and *Support*. For the sake of brevity, the attributes associated with each group will not be discussed in detail; only the purpose of each group will be presented. For a more detailed view of the group attributes refer to [2] and to Appendix B.

Four groups, *ComponentID*, *Software Component Information*, *Software Signature*, and *Verification*, are used to identify and verify a software component. These groups describe the manufacturer, the specific product configuration, the means with which to detect the product's existence, and the product's level of operation, respectively.

The *Location* group is used to describe locations where files and components reside. Other groups reference an index into the location group rather than specifying the location directly; this is intended to simplify management by separating out location information. The *Equivalence* group describes other software components that are equivalent (in some unspecified manner) to the one being described; the intended use of this group is unclear. The *Superseded Components* group describes products that have been superseded by the software component being described in the specification.

The *Maintenance* group is used only when the software component being described is actually a maintenance release or patch to another software component and describes the component to which the maintenance is to be applied.

The *Subcomponents* group is used to support "suite" style software components (e.g., Microsoft Office). The *Component Dependencies* group identifies the software components on which the described software component depends, while the *Attribute Dependencies* group specifies the value of specific attributes on which the software component being described depends on in order to operate correctly.

The *File List* group lists the files that comprise the software component being described. The *Installation* group specifies the available installation and de-installation processes for the software component. The *Installation Log Files* group specifies the names, locations, and descriptions of log files created during the installation process. Lastly, the *Support* group gives information used to obtain product support for the described software component.

4.2. Evaluation

The Software MIF from the DMTF Application Management working group has a longer history, and hence is more well-defined, than the OSD specification effort. The major contribution of the DMTF is the collection of schema definitions from the various working groups. Specifically related to software deployment, the Software MIF has defined the above schema and many standard values for the attributes contained in that schema. This type of standardization is necessary to provide any type of general solution to the software deployment dilemma. As a result, companies are releasing computing systems that conform to various pieces of the Software MIF standard. Wide-scale adoption has still not occurred and only basic capabilities are provided such as inventory management. Currently the Software MIF only addresses installation, update, and de-installation from the software deployment life cycle, while the adapt process receives limited support in the form of operational assert constraints.

A single Software MIF specification describes a single version and variant of one component. One very important aspect about the Software MIF is that it is not "Web enabled" in the sense of using any World Wide Web standards or protocols. Some of the Software MIF characteristics are detailed below.

Consumer site description: The complete definition of the DMTF effort specifically includes a client-side model through its DMI interface. DMTF has established working groups to create standard schema to describe various hardware configurations as well as the software description discussed here.

Assert constraints: Assert constraints are supported in the form of attribute dependencies. A dependency can be placed on any attribute defined in the standard DMTF schema definitions. Asserts are used to select a properly configured software system to install, and through the consumer site model they possibly could be used to enforce operational correctness, though this is not a main focus.

Dependency constraints: Software MIF dependencies are not flexible with respect to defining a dependency constraint. It is always the case that a specific component version (or range of versions) is considered to be a dependency constraint. This implies that a dependency is always on a versioned subsystem, and this is likely to be too simplistic. There is no support for dependency constraints based on something other than a subsystem or on the need of a different configuration or functional capability. For example, a software system may be dependent upon a particular configuration of a dependent subsystem, but there is no way this can be specified.

No clear support exists for the notion of a generic or capability dependency such as the generic dependency on an HTML viewer. The inclusion of the Equivalence group in the Software MIF schema may have some intended use in this area, but it is not clear how useful it is since it requires the creation of an explicit list of equivalences.

Artifacts: The Software MIF provides the necessary support for artifact description by providing a place to specify each software artifact that comprises the software system and its location and type.

Configuration and Activities: There is no support for configurable software systems in the Software MIF. It is either assumed that the component description is all or nothing, or that any configurability is handled externally. In addition, the Software MIF is largely devoid of any process or activity related description. At a very high level, the Software MIF supports installation and de-installation processing by providing a place in the schema to point to an installation and de-installation program, respectively. This can be considered a high-level process description, but it is not very useful. This approach does not allow general interpretation or reasoning about the processing steps or activities that need to be performed, which in turn disallows the creation of a generic procedure for handling these processes.

5. Conclusion

The use of networks, such as the Internet, to distribute software to consumers is proving to be very beneficial for both the consumer as well as the software producer. In order to realize the potential of network software distribution, certain key areas of technology must be created and introduced. One such technology is the definition and standardization of software system and component description. Describing software systems and components in a complete and rigorous manner is required to enable and create a general infrastructure to support the software deployment life cycle. Such a software description definition must include ways to describe system assert constraints, dependency constraints, artifacts, configurations, and specialized deployment activities. Combining such a definition with a semantic description of consumer sites makes it possible to create general solutions to the various deployment tasks.

Specific efforts, such as Microsoft's and Marimba's OSD and DMTF's Software MIF, are attempting to address the need for software system and component description to varying degrees of success and detail. These efforts, while generally steps in the right direction, still lack the level of sophistication required to fully support the software deployment life cycle. This paper has outlined the specific requirements that are needed by software system and component description techniques and has evaluated the ability of OSD and MIF to address those needs.

Research work that is directly related to this topic is being performed by the authors in the form of the Software Dock [4]. The Software Dock is a distributed, agent-based framework for supporting the entire software deployment life cycle. One major aspect of the Software Dock research is the creation of a standard schema to meet the requirements outlined in this paper.

The current prototype of the Software Dock system includes an evolving software description schema definition that is comprised of elements for describing assertions, dependencies, artifacts, and configuration information. Abstractly, the Software Dock provides infrastructure for housing software releases and their semantic descriptions at release sites and provides infrastructure to deploy or "dock" software releases at consumer sites. Mobile agents are provided to interpret the semantic descriptions provided by the release site in order to perform various software deployment life cycle processes.

Initial implementations of generic agents for performing configurable content install, update, adapt, reconfigure, and remove have been created. These agents are completely parameterized by the information provided in the semantic descriptions of the software releases. For more information regarding the Software Dock and other research being performed within the Software Engineering Research Laboratory, refer to *http://www.cs.colorado.edu/serl*.

Acknowledgements

This materiel is based upon work sponsored by the Air Force Materiel Command, Rome Laboratory, and the Defense Advanced Research Projects Agency under Contract Numbers F30602-94-C-0253 and F30602-98-2-0163. The content of the information does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred.

References

[1] T. Bray. *"Extensible Markup Lanuage (XML): Part I. Syntax"*. Textuality, Vancouver, BC, Canada. (http://www.w3.org/pub/WWW/TR/WD-xml-lang.html).

[2] Desktop Management Task Force. *"Software Standard Groups Definition, Version 2.0"*, 29 Nov. 1995. (http://www.dmtf.org/tech/apps.html).

[3] C. Ellerman. *"Channel Definition Format"*. Microsoft Corp, Redmond, WA. (http://www.w3.org/TR/NOTE-CDFsubmit.html).

[4] R. S. Hall, D. Heimbigner, A. van der Hoek, and A. L. Wolf. "An architecture for Post-Development Configuration Management in a Wide-Area Network". In *Proc. of the 1997 Intl. Conf. on Distributed Computing Systems*, pages 269–278. IEEE Computing Society, May 1997.

[5] R. S. Hall, D. Heimbigner, and A. L. Wolf. "Software Deployment Languages and Schema". Technical Report CU-SERL-203-97, University of Colorado Software Engineering Research Laboratory, 18 Dec. 1997. (http://www.cs.colorado.edu/serl/cm/Papers.html#Schema).

[6] R. S. Hall, D. Heimbigner, and A. L. Wolf. "Requirements for Software Deployment Languages and Schema". In *Proc. of the 8th Intl. Symposium on System Configuration Management (SCM-8)*, Brussels, July 1998.

[7] Microsoft Corp, Redmond, WA. *"Zero Administration Initiative"*. (http://www.microsoft.com/windows/innovation/).

[8] A. van Hoff, H. Partovi, and T. Thai. *"The Open Software Description Format (OSD)"*. Microsoft Corp. and Marimba, Inc. (http://www.w3.org/TR/NOTE-OSD.html).

Appendix A - OSD example

This example is taken directly from the OSD specification for W3C [8].

```
<SOFTPKG NAME="com.foobar.www.Solitaire"
   VERSION="1,0,0,0">
   <TITLE>Solitaire</TITLE>
   <ABSTRACT>Solitaire by FooBar Corporation</ABSTRACT>
   <LICENSE HREF=
      "http://www.foobar.com/solitaire/license.html"/>
   <!--FooBar Solitaire is implemented in native code
       for Win32, Java code for other platforms -->

   <IMPLEMENTATION>
      <OS VALUE="WinNT">
         <OSVERSION VALUE="4,0,0,0"/></OS>
      <OS VALUE="Win95"/>
      <PROCESSOR VALUE="x86"/>
      <LANGUAGE VALUE="en"/>
      <CODEBASE HREF=
         "http://www.foobar.org/solitaire.cab"/>
   </IMPLEMENTATION>

   <IMPLEMENTATION>
      <IMPLTYPE VALUE="Java"/>
      <CODEBASE HREF=
         "http://www.foobar.org/solitaire.jar"/>
      <!-- The Java implementation needs the
           DeckOfCards object -->
      <DEPENDENCY>
         <CODEBASE HREF=
            "http://www.foobar.org/cards.osd"/>
      </DEPENDENCY>
   </IMPLEMENTATION>
</SOFTPKG>
```

Appendix B - MIF example

The MIF example tries to mimic the OSD example where possible. MIF is verbose so many details have been left out at the places marked by the ellipsis symbol. In the case of attributes, most details have been omitted, such as identification number, description (where deemed unnecessary), access rights, storage type, and value. In addition, other attribute groups have been entirely omitted as being irrelevant. These include the software signature, equivalence, verification, subcomponents, superceded components, installation log file, and support.]

```
START COMPONENT
    Name = "ComponentID"
    Class = "DMTF|ComponentID|001"
    START ATTRIBUTE
        Name = "Manufacturer"
        ID = 1
        Description = "Manufacturer of this component"
        Access Read-Only
        Storage = Common
        Type = String(64)
        Value = "FooBar Corporation"
    END ATTRIBUTE
    START ATTRIBUTE
        Name = "Product"
        ID = 2
        Description = "Product name of this component"
        Access Read-Only
        Storage = Common
        Type = String(64)
        Value = "Solitaire"
    END ATTRIBUTE
    START ATTRIBUTE
        Name = "Version"
        ...
        Value = "1.0"
    END ATTRIBUTE
    START ATTRIBUTE
        Name = "Serial Number"
        ...
        Value = "1234567890"
    END ATTRIBUTE
    START ATTRIBUTE
        Name = "Installation"
        Description = "Automatically assigned date"
        ...
        Value = " "
    END ATTRIBUTE
    START ATTRIBUTE
        Name = "Verify"
        Description = "Level of verification"
        Type = START ENUM
            0 = "An error occurred; check status code"
            1 = "This component does not exist"
            2 = "The verify is not supported"
            3 = "Reserved"
            4 = "This component exists, but the"
                "functionality is untested"
            5 = "This component exists, but the"
                "functionality is unknown"
            6 = "This component exists, and is not"
                "functioning correctly"
            7 = "This component exists, and is"
                "functioning correctly"
        END ENUM
        ...
        Value = 2
    END ATTRIBUTE
END GROUP
BEGIN GROUP
    Name = "Software Component Information"
    Class = "DMTF|Software Component Information|002"
    Description = "This group contains additional"
                  "identifying information"
    ...
    START ATTRIBUTE
        Name = "Target Operating System"
        ...
        Type = START ENUM
            0 = "Other"
            1 = "DOS"
            2 = "MACOS"
            3 = "OS2"
            4 = "UNIX"
            5 = "WIN16"
            6 = "WIN32"
            7 = "OPENVMS"
            8 = "NetWare"
        END ENUM
        Value = 6
    END ATTRIBUTE
    START ATTRIBUTE
        Name = "Language Edition"
        ...
        Value = "en"
    END ATTRIBUTE
    ...
END GROUP
START GROUP
    Name = "Software Location"
    Class = "DMTF|Software Location|001"
    Description = "This group identifies the various"
                  "loca-
tions where parts of the component"
                  "have been installed"
    START ATTRIBUTE
        Name = "Index"
        ...
        Type = Integer
    END ATTRIBUTE
    START ATTRIBUTE
        Name = "Type"
        ...
        Type = START ENUM
            0 = "Unknown"
            1 = "Other"
            2 = "Product base directory"
            3 = "Product executables directory"
            4 = "Product library directory"
            5 = "Product configuration directory"
            6 = "Product include directory"
            7 = "Product working directory"
            8 = "Product log directory"
            9 = "Shared base directory"
            10 = "Shared executables directory"
            11 = "Shared library directory"
            12 = "Shared include directory"
            13 = "System base directory"
            14 = "System executables directory"
            15 = "System library directory"
            16 = "System configuration directory"
            17 = "System include directory"
            18 = "System log directory"
        END ENUM
    END ATTRIBUTE
    START ATTRIBUTE
        Name = "Path"
        ...
    END ATTRIBUTE
END GROUP
START TABLE
    Name = "Software Location"
    Class = "DMTF|Software Location|001"
    { 1, 2, "C:\\SOLITAIRE" }
    { 2, 3, "C:\\SOLITAIRE\\BIN" }
    { 3, 4, "C:\\SOLITAIRE\\DLL" }
END TABLE
START GROUP
    Name = "Component Dependencies"
    Class = "DMTF|Component Dependencies|001"
    Description = "This group identifies dependen-
cies this"
                  "component has on other components"
```

```
    START ATTRIBUTE
        Name = "Manufacturer"
        ...
    END ATTRIBUTE
    START ATTRIBUTE
        Name = "Product"
        ...
    END ATTRIBUTE
    START ATTRIBUTE
        Name = "Minimum Version"
        ...
    END ATTRIBUTE
    START ATTRIBUTE
        Name = "Maximum Version"
        ...
    END ATTRIBUTE
    START ATTRIBUTE
        Name = "Type"
        ...
        Type = START ENUM
            0 = "Exclude"
            1 = "Required"
            2 = "One of a set required"
        END ENUM
    END ATTRIBUTE
    START ATTRIBUTE
        Name = "Set Index"
        ...
    END ATTRIBUTE
END GROUP
START TABLE
    Name = "Component Dependencies"
    Class = "DMTF|Component Dependencies|001"
    { "Foobar Corpora-
tion", "Cards", "2.1", "3.0", 1, 0 }
END TABLE
START GROUP
    Name = "Attribute Dependencies"
    Class = "DMTF|Attribute Dependencies|001"
    Description = "This group identifies dependen-
cies on"
                  "attribute values"
    START ATTRIBUTE
        Name = "Class"
        ...
    END ATTRIBUTE
    START ATTRIBUTE
        Name = "ID"
        ...
    END ATTRIBUTE
    START ATTRIBUTE
        Name = "Minimum Value"
        ...
    END ATTRIBUTE
    START ATTRIBUTE
        Name = "Maximum Value"
        ...
    END ATTRIBUTE
    START ATTRIBUTE
        Name = "Type"
        ...
        Type = START ENUM
            0 = "Exclude"
            1 = "Required"
            2 = "One of a set required"
        END ENUM
    END ATTRIBUTE
    START ATTRIBUTE
        Name = "Set Index"
        ...
    END ATTRIBUTE
END GROUP
START TABLE
    Name = "Attribute Dependencies"
    Class = "DMTF|Attribute Dependencies|001"
    { "DMTF|Video|001", 6, "VGA", "VGA", 2, 1 }
    { "DMTF|Video|001", 6, "SVGA", "SVGA", 2, 1 }
END TABLE
BEGIN GROUP
```

```
    Name = "File List"
    Class = "DMTF|File List|001"
    Description = "This group describes the files"
                  "included in the software product"
    START ATTRIBUTE
        Name = "Location"
        Description = "Index into the location table"
        ...
    END ATTRIBUTE
    START ATTRIBUTE
        Name = "Name"
        ...
    END ATTRIBUTE
    START ATTRIBUTE
        Name = "Description"
        ...
    END ATTRIBUTE
END GROUP
START TABLE
    Name = "File List"
    Class = "DMTF|File List|001"
    { 3, "solitaire.exe",
      "Primary executable for this application" }
    { 3, "install.exe",
      "Configuration executable for this"
      "application" }
    { 4, "solitaire.dll",
      "Library routines for this application" }
    { 2, "readme.txt", "Readme notes for this"
      "application" }
END TABLE
START GROUP
    Name = "Installation"
    Class = "DMTF|Installation|001"
    Description = "This group contains information to"
                  "allow install, update, and deinstall"
                  "of this software component."
    START ATTRIBUTE
        Name = "Type"
        ...
        Type = START ENUM
            0 = "Other"
            1 = "Install"
            2 = "De-install"
        END ENUM
    END ATTRIBUTE
    START ATTRIBUTE
        Name = "Name"
        ...
    END ATTRIBUTE
    START ATTRIBUTE
        Name = "Description"
        ...
    END ATTRIBUTE
    START ATTRIBUTE
        Name = "Install size"
        ...
    END ATTRIBUTE
    START ATTRIBUTE
        Name = "Command Line"
        ...
    END ATTRIBUTE
    START ATTRIBUTE
        Name = "Location"
        ...
    END ATTRIBUTE
END GROUP
START TABLE
    Name = "Installation"
    Class = "DMTF|Installation|001"
    { 1, "Deinstall", "Uninstalls this product",
      359.8, "install.exe /d", 5 }
END TABLE
```