# *Towards a CIM Schema for RunTime Application Management*

Alexander Keller, Heather Kreger, Karl Schopmeyer

*IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598, USA, alexk@us.ibm.com*
*IBM Corporation, P.O. Box 12195, Research Triangle Park, NC 27709, USA, kreger@us.ibm.com*
*The OpenGroup, k.schopmeyer@opengroup.org*

We describe new extensions to the Core, Application and System Schemas of the Common Information Model (CIM). While previous work on the CIM Application Schema mainly dealt with the distribution, deployment and installation aspects of software, a recent effort within the Distributed Management Task Force (DMTF) is targeted at defining schema extensions capable of dealing with the full lifecycle of distributed applications, especially the runtime stage. These application runtime extensions are currently developed by the DMTF Application Working Group in which the authors actively participate. In order to serve as a basis for a runtime model of distributed applications, various extensions had to be applied to the existing schemas. These schema extensions, described in this paper, have been adopted by the CIM Technical Committee and are part of the recently released version 2.5 of the CIM schema.

**Keywords:** Common Information Model, Runtime Application Management, Application Schema

## 1 Introduction

With over 700 object classes, a model for interoperability and the completetion of the event model, the acceptance of the Common Information Model (CIM) [1, 2, 8] by leading vendors of devices, systems and network components is gaining momentum throughout the industry. However, despite a large amount of work on the CIM Application Schema [3], CIM has not yet been widely deployed for managing distributed applications. One reason is that the current version of the Application Schema mainly focuses on the software deployment, distribution and installation aspects of applications, but does not address the issue of monitoring and managing applications at runtime, i.e., once they get instantiated. On the other hand, the need for a standardized model, capable of managing distributed applications and services throughout their complete lifecycle becomes increasingly important in today's distributed environments [9], where *Application Service Providers (ASP)* offer managed services and applications to users on the Internet.

This paper describes the efforts of the DMTF Application Working Group in which the authors participate, that aim at extending the existing CIM Application Schema so that it becomes a solid basis for managing distributed applications throughout their lifecycle and – in particular – at runtime. An important prerequisite for this is to ensure that the CIM Core, System and Application schemas are able to express the various types of distributed applications and to enhance the schemas, where needed. The goal of our work is to reuse the existing schemas as much as possible and to make the new extensions seamlessly fit into the existing framework. The paper describes our experiences with extending the CIM schemas and discusses the reasons behind our design.

The paper is structured as follows: Section 2 briefly states the requirements for CIM-based application management. Section 3 introduces the core modeling principles of CIM and describes the CIM Application Schema and its most important classes, the main targets of our effort. The schema extensions, developed by the working group and adopted by the CIM Technical Committee for publication in CIM 2.5, are described in detail in section 4. Section 5 concludes the paper by discussing the lessons we learned during the design of the schema extensions and presents current work items of the group.

*O. Festor and A. Pras (Eds.) : 12th International Worshop on Distributed Systems: Operations and Management*
*DSOM'2001 Nancy France, October 15-17, 2001.*

# 2 Requirements for CIM-based Application Management

In today's distributed environments, applications provide the business functions directly and form the basis for much of the technical function of our information technology (IT) systems. Therefore, applications bring us closer to the real interests of the users, the services delivered and the business systems implemented: Clearly, the goal of IT is to deliver services and to provide effective implementation of business functions. Applications are the embodiment of these services and business functions. Thus, the goals of application management incorporate not only the management of the individual applications themselves but of the services and business functions they provide. One should require a minimal knowledge of the application function in order to provide generalized management of the application. Mapping a business service offered to customers to its implementation (i.e., the products used) and mapping the implementation to the running instance is crucial for the management of large-scale distributed applications. An information model for applications must be able to express the various logical and physical representations of a distributed application. In addition, it should provide navigation facilities by means of associations.

That said, an information model must be able to unambiguously identify multiple installations of the same software product on a system. In addition, it must be guaranteed that multiple instances of the same software element can be distinguished the one from the other. A similar situation (later in the software lifecycle) occurs if many users run the same software simultaneously, i.e., the same software element has multiple instances. The CIM Application Schema must be able to distinguish between these instances despite the fact that applications are very diverse in form, structure and organization: Application structure covers everything from simple scripts to complex structures of dynamically loadable classes. The dynamics of applications appear in several ways:

- they typically change more rapidly than the other components of the IT environment,
- they are often used in a temporary manner (loaded, unloaded, restarted many times),
- they are often created from components that are dynamically bound together.

Finally, it should be recognized that the form and structure of applications is changing today: With the advent of the Internet, component-based development, dynamic server environments, and service-oriented architectures, the structure, organization and operation of applications is changing rapidly today. Extensions to the CIM Application Schema must consider these emerging architectures.

# 3 Existing CIM Work related to Application Management

This section gives a brief overview of the CIM schemas (section 3.1) and modeling principles (section 3.2) and describes the current version of the Application Schema in section 3.3. Although it is not our goal to provide a tutorial on CIM within the scope of this paper, the discussion of our work and our resulting design choices in the following sections are better appreciated if the implications of the underlying modeling principles and schemas are well understood.

## *3.1 Overview of the CIM Schemas*

CIM follows an object-oriented approach and defines managed resources as object classes with properties and methods that can be further refined by means of strict inheritance. In order to circumvent multiple inheritance, CIM makes extensive use of relationships, namely associations and aggregations; both are modeled as association classes (see [2]). CIM uses the *Unified Modeling Language (UML)* [10] as notation for specifying the various schemas, thus leveraging existing general-purpose development and software engineering tools. It is therefore possible to transform the managed object classes – derived from the Core and Common Schemas – into widely used programming and description languages (OMG IDL, C++, Java, XML). The CIM syntax for management information, the *Managed Object Format (MOF)*, however, is

specific to CIM and thus not supported by the code generators of off-the-shelf CASE tools. In addition, CIM introduces several extensions to the UML, which are thus unsupported in current CASE tools, but crucial for the proper functioning of a CIM based management system. The nature and purpose of these extensions is described in section 3.2.

The *Core Schema* defines basic terms as abstract classes such as service access point, service, product, system, logical element etc., and provides a means for associating context (e.g., setting, configuration) with them. It is the basis for the various common schemas; currently, CIM comprises more that 15 different common schemas that relate to various resource types. An excellent detailed description of the CIM Core Schema and the modeling principles can be found in [4].

The *System Schema* is one of the various Common Schemas and refines the root classes of the Core Schema in order to deal with jobs, hosts, operating systems, processes, threads and file systems.

The *Application Schema*, the Common Schema that we are most interested in for our work, refines the Core Schema with respect to software packages and applications. We will discuss it in greater detail in section 3.3.

Finally, the *Distributed Application Performance Schema (DAP)* relates the definition, the metrics and the logical element that is instantiated to a so-called "unit of work". The DAP schema provides a means to run synthetic transactions against a distributed application and to measure its response time.

In total, the amount of managed object classes defined in CIM comes close to 750. It is therefore fair to say that even if CIM is still evolving, it represents a solid basis for integrated management. The exchange of management information between managed resources, management systems and management applications is done by encoding the CIM object descriptions in XML according to [6] and executing CIM operations over HTTP, as specified in [5]. A white paper [7] describes the details on using XML for representing management information in CIM.

## *3.2 CIM Extensions to UML*

We will now describe some of the core modeling principles of CIM that go beyond the UML and thus need to be taken into account when extending the CIM schemas.

### *3.2.1 Qualifiers*

Qualifiers are the way how meta-data is added to all artifacts of CIM, namely classes, properties, methods, associations and aggregations. CIM distinguishes three types of qualifiers (a detailed description is given in [2]):

- **Meta Qualifiers** describe general-purpose and syntactical elements, such as whether a MOF element is a description, or whether a class represents an association (recall that relationships are modeled as classes in CIM).

- **Standard Qualifiers** add additional semantics to the elements of a MOF description, such as whether a class is abstract, an association is an aggregation, or if the property has been defined in the scope of another management architecture (such as the *IETF Structure of Management Information* or the DMTF *Distributed Management Interface (DMI)*) and can be retrieved from an appropriate translator ("mappingstrings"). Other examples of standard qualifiers are "key" (further described in section 3.2.2) or "weak" and "propagated" (see the discussion in section 3.2.3).

- **Optional Qualifiers** contain useful information for management applications, such as whether an operation requires a considerable amount of computation time ("expensive") and space ("large"), or whether an element is not supposed to be displayed by a management console because it is used for internal purposes only ("invisible").

### 3.2.2 CIM Naming: Keys

In CIM, keys are the mechanism for uniquely naming and identifying managed objects and their associations. The properties of a managed object class that carry the "key" qualifier are used for constructing the key of an object; typical examples of such properties are "Name", "Version" or "ID". In case these properties are not sufficient to uniquely identify an instance, further properties need to be added to the key, such as the property "SoftwareElementState" for uniquely identifying instances of the class *CIM_SoftwareElement* in the CIM Application Schema. In order to further distinguish instances whose different classes have a common superclass, the key-qualified property "CreationClassName" is added.

Associations (and aggregations) are identified by having at least two properties that carry the references to the (two) associated objects. Since associations are objects themselves, they may carry further information relating to the association itself in additional properties.

As a corollary to the foregoing descriptions, it is important to note that all concrete managed object classes in CIM *must* inherit a key structure and *must not* change it.

### 3.2.3 "Weak" Associations and Propagated Keys

"Weak" associations (identified by a "w" within schema diagrams) are used in CIM to express that an object does not exist on its own and, if instantiated, is subject to the lifecycle of another object. "Weak" objects exist only within the scope of an instance of exactly one associated class. The cardinality of the association on the side of this associated class must always be "1" because any different cardinality would imply that the "weak" object instance(s) would have to adopt keys from multiple objects (or no keys at all), which clearly does not make sense. Typical examples of "weak" objects are instances of *CIM_File* that can only exist when the *CIM_Filesystem* class is instantiated. *CIM_Filesystem*, on the other hand, is weak to *CIM_ComputerSystem* because file systems are hosted by computer systems and cannot exist as "standalone" objects. The managed object classes of "weak" objects therefore need to provide additional scoping information, which is done in CIM by propagating the keys of the class on the other side of a "weak" association to them. Such "propagated keys" carry the PROPAGATED qualifier and a reference to the property of the object class from which the key stems. Consequently, it is neither possible to change propagated keys, nor to modify the inheritance relationships of a "weak" class (i.e., moving it under another class in the model). Ignoring weak associations (e.g., when implementing a CIM provider) implies that CIM-based management applications will not be able to find and address the objects offered by a CIM provider because the keys will not match.

## 3.3 The CIM Application Schema

Since our work mainly deals with extending the CIM Application Schema, we will now describe its main object classes and associations. The schema is depicted in figure 1.

*CIM_ApplicationSystem* is oriented towards a business function of an application and helps to group its various components. Typical examples for application systems are: "e-business storefront application", "database system" or "word processor". Note that *CIM_ApplicationSystem* is able to aggregate various other *CIM_ManagedSystemElement*s (and, thus, can contain other *CIM_ApplicationSystem*s) because it is subclassed from *CIM_System*, which is related to *CIM_ManagedSystemElement* by means of the aggregation *CIM_SystemComponent*. We can therefore express that an "e-business storefront application" comprises e.g., a "database system". In addition, *CIM_ApplicationSystem* collects software features (see below) from one or more products to fulfill a business function.

*CIM_Product*, defined in the Core Schema (and not depicted in Figure 1), deals with defining the units of acquisition and contains information related to licensing, support and warranty. An example of a *CIM_Product* is "DB2 Universal Database v7.1".

*CIM_SoftwareFeature* defines a particular function or capability of a *CIM_ApplicationSystem* or *CIM_Product* that is meaningful to a customer or a user. It reflects functions or roles of an application

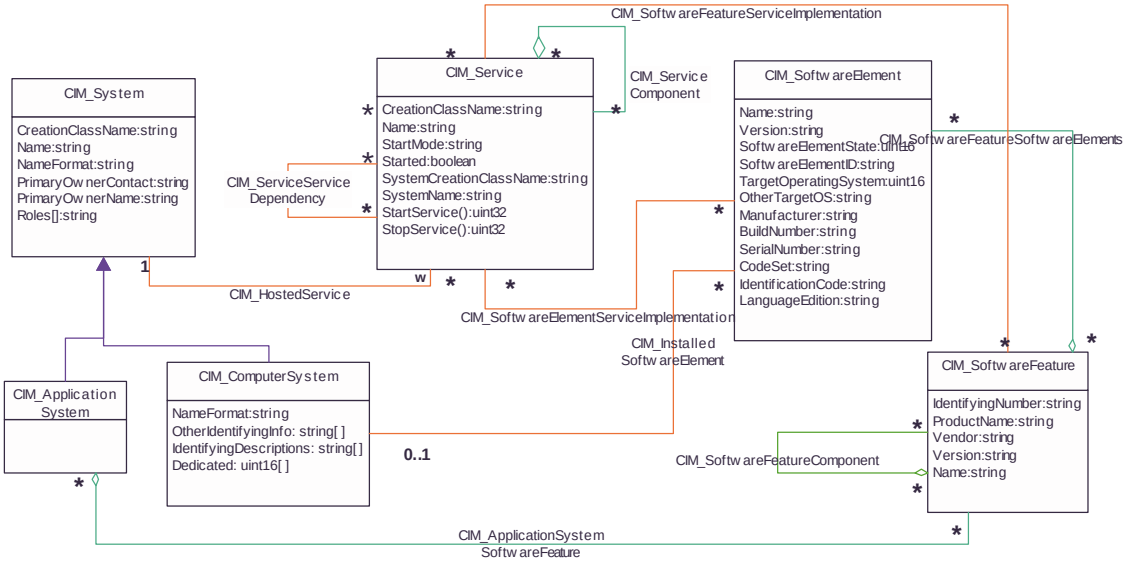


**Fig. 1:** Relevant Classes of the CIM Application Schema (without *CIM_Check* and *CIM_Action*)

component and defines the units of component management. For example, the *CIM_ApplicationSystem* "database system" consists of various parts that are meaningful to a user: DB2 Engine, DB2 Runtime Environment, DB2 JDBC Driver, DB2 Replication, DB2 Parallel Extension, DB2 Administration Server are all examples of *CIM_SoftwareFeatures*. The aggregation *CIM_ApplicationSystemSoftwareFeature* groups instances of these *CIM_SoftwareFeatures* under a specific instance of *CIM_ApplicationSystem*. In the concrete case of DB2, the features mentioned above are also the "high-level" parts of the database product, meaning that the aggregation *CIM_ProductSoftwareFeatures* ties instances of *CIM_SoftwareFeature* and *CIM_Product* together.

*CIM_SoftwareElement*s are the most fined-grained objects in the Application Schema; they represent units of deployment, such as collections of files and associated details being individually managed. Examples of *CIM_SoftwareElement* are the various files that are collected by means of the aggregation *CIM_SoftwareFeatureSoftwareElements* under an individual *CIM_SoftwareFeature*, such as configuration files, scripts, binaries or (shared) libraries. *CIM_SoftwareElement* provides information indicating the compatibility with a specific operating system in the "TargetOperatingSystem" property. In addition, *CIM_SoftwareElement* is the only place where information relating to the state (Deployable, Installable, Executable, Running) of a software component is kept in CIM. This implies that information regarding the software lifecycle is confined to these fine-grained deployment units; the state of the overall application system or its features is not captured in CIM. In addition, the "SoftwareElementState" property is part of the key (see section 3.2.2) of *CIM_SoftwareElement*, which implies there can be multiple instances of the same *CIM_SoftwareElement* simultaneously (distinguished by a different value of "SoftwareElementState"). Thus, one can neither assume that the value of the "SoftwareElementState" property is determined by an underlying state machine nor a sequential order of a *CIM_SoftwareElement*'s states. Finally, note that the values of the "SoftwareElementState" property reflect *deployment* states (although the value "Running" seems to imply operational data); no information regarding the *operational* state is kept in this property (cf. the discussion on the "Status" property of *CIM_ManagedSystemElement* in section 4.2.5).

It should be noted that there are no relationships between *CIM_SoftwareElement* and *CIM_Product* or *CIM_ApplicationSystem*. If a query is issued to a CIM Object Manager (CIMOM) for enumerating the files that make up a *CIM_Product*, a CIMOM would first need to perform the intermediate step of obtaining the products' *CIM_SoftwareFeature*s, and then navigate the *CIM_SoftwareFeatureSoftwareElement* aggregation to determine the *CIM_SoftwareElements*.

*CIM_Service*, defined in the Core Schema, represents the conceptual service being provided by either a *CIM_SoftwareFeature* or a *CIM_SoftwareElement*. We have chosen to provide examples for the former case, i.e., we relate *CIM_Service* to a *CIM_SoftwareFeature*, which yields Services such as "DB2 Replication Service", "DB2 Admin Service" etc. Note the "weak" association *CIM_HostedService* between *CIM_Service* and *CIM_System*, which implies that a service can only exist within the scope of a system. The impact of this "weak" association on our work is discussed in section 4.1.

*CIM_ServiceAccessPoint* is the programmatic interface (or the port) to a *CIM_Service*. In our specific example, the SAP of "database system" is encapsulated by the ODBC/JDBC driver.

Finally, the Application Schema contains the classes *CIM_Action* and *CIM_Check* (not depicted in figure 1). Derived from them are more than a dozen subclasses that define various generic actions (Reboot, ModifySetting) and checks (OSVersion, VersionCompatibility etc.) relating to the deployment and installation of *CIM_SoftwareElement*s. As we will discuss in section 4.1, their focus on *CIM_SoftwareElement* by means of "weak" associations prevents the applicability of *CIM_Action* and *CIM_Check* to other classes.

It is evident from the foregoing descriptions that at its current stage, the CIM Application Schema is restricted to the deployment and installation aspects of software packages, a task which is usually confined to a single host. There is no support yet for addressing the runtime characteristics of a distributed application. The next section focuses on the initial schema extensions to provide support for the runtime stage of applications that can be distributed among multiple computer systems.

# 4 Extending the CIM Schemas

We will now discuss the implications of the existing schema structure on our modeling effort (section 4.1) and, in section 4.2, the design decisions behind the extensions that have been included in version 2.5 of CIM, released in february 2001.

## *4.1 Initial modeling Efforts*

CIM is supposed to be deployed by reusing the existing schemas as much as possible and extending them both by adding new subclasses and associations, together with new properties and methods. This should ensure that all the requirements are met while keeping the impact on existing schemas as small as possible. However, the following examples show that the use of keys and "weak" associations in the current version of the CIM schemas pose severe constraints and make some desired changes impossible without modifying the key structures (and thus requiring a version change).

It seems natural to define the classes for introducing the concept of dynamic, user-defined operations either by extending *CIM_Action* (Application Schema) or subclassing from it. However, *CIM_Action* is defined as "weak" to *CIM_SoftwareElement*, which implies that any extensions can only be applied to the objects of finest granularity that make up an application. Thus, it is not possible to associate *CIM_Action* or any of its subclasses to any other class than *CIM_SoftwareElement*. Consequently, classes representing dynamic operations therefore would have to be defined separately from *CIM_Action* and could not reuse any functionality already defined in *CIM_Action* or any of its already existing subclasses. The same problem is encountered when trying to define preconditions on operations for which *CIM_Check* and its subclasses would seem a natural fit. However, *CIM_Check*, too, is weak to *CIM_SoftwareElement*, which precludes its reuse for any other classes (such as *CIM_SoftwareFeature* or *CIM_ApplicationSystem*). Since a mechanism for defining generic operations and their preconditions on any other classes than *CIM_SoftwareElement* cannot reuse existing classes, work on a schema supporting the definition of dynamic, user-defined operations has been postponed to a subsequent version of the CIM schemas.

Another thought was that *CIM_Service* could be taken as the basis for modeling IT services spanning multiple systems. However, the fact that *CIM_Service* is weak to *CIM_System* (meaning that services exist only within the scope of a – single – system and therefore are not allowed to exist after the underlying *CIM_System* has been stopped) prohibits such a straightforward extension. In ad-

dition, *CIM_Service* is supposed to represent the "running" or "executing" representation of either a *CIM_SoftwareElement* or a *CIM_SoftwareFeature* (for a given *CIM_Service*, only one type of the associations *CIM_SoftwareFeatureServiceImplementation* or *CIM_SoftwareElementServiceImplementation* may be instantiated at a time), thus having different semantics than an end-to-end service implemented by several (distributed) software components. Another problem is that *CIM_Service* does not have a property "InstanceID" that acts as a key; one therefore cannot distinguish between multiple service instances on the same system. This precludes the distinction between, e.g., several http daemons. On the other hand, adding a new key to *CIM_Service* breaks existing CIM implementations because many classes in the various common (and the derived resource-specific) schemas inherit from *CIM_Service*.

In order to capture the bindings of an application through its lifecycle, the idea to reuse *CIM_Dependency* for modeling start/stop/failure dependency relationships comes to mind. This would allow the specification of relationships indicating, e.g., "Application X must be running/terminated before Application Y can be started/stopped" or "Application X acts as a failover (i.e., backup) for Application Y". However, the issue we faced when trying to add properties to *CIM_Dependency* that reflect these various dependency types is that many classes in the various common (and the derived resource-specific) schemas inherit from *CIM_Dependency*, for which the notion of dependency types does not make sense. A typical example is *CIM_AssociatedBattery*, defined in the Device Schema, which associates one (or more) *CIM_Battery* object(s) with a *CIM_LogicalDevice*. Another reason why we could not introduce a subclass of *CIM_Dependency* that fits our purposes is that more than one dependency type may be defined between the same objects, which results in multiple instances of the same class (with different property values). Since the new property "dependencyType" cannot be part of the key, different dependency objects would share the same key. Finally, new types are introduced in CIM by subclassing while our approach would imply to distinguish between types by means of (enumerated) properties. Adopting the latter would have led to inconsistencies in the way how CIM modeling is done.

## 4.2 New Schema Extensions for managing Applications in CIM 2.5

In this section, we discuss the recently adopted schema extensions that relate to the management of applications. These extensions have been worked out by the CIM Application Working Group and adopted by the CIM Technical Committee to be included in CIM 2.5. For each item, we will describe the nature of the extensions and discuss the reasons for our design decisions.

### 4.2.1 Associate Services with Processes

The *CIM_ServiceProcess* association, defined in the CIM System Schema, is used to establish and navigate relationships between services and system processes. It allows a management application to determine which process represents a specific running application instance and vice-versa. Figure 2 depicts the details of *CIM_ServiceProcess*. Because associations are modeled as classes in CIM, it is therefore possible to add a property `ExecutionType` that provides detailed information in which configuration a service is running and reflects two possible types of application topology.

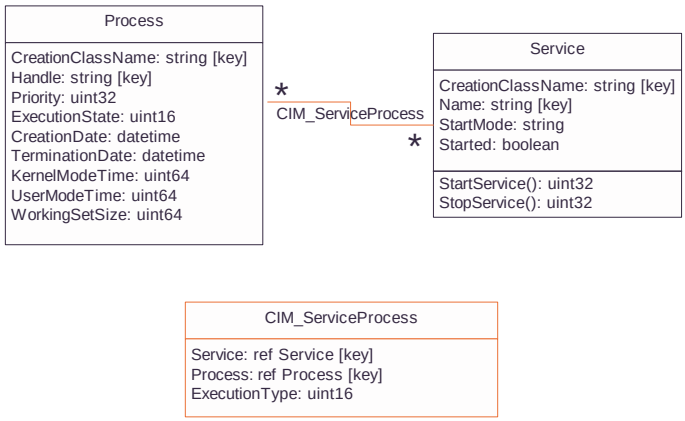


**Fig. 2:** The *CIM_ServiceProcess* association

There are two possibilities: The service can run on its own, thus independently from any other application, (value: "Executes as Independent Process"), i.e., the service is responsible for the lifecycle of the process.

This implies a 1:1 mapping between a service and the process(es). Another possibility is that the service runs within an existing process and is not visible in the process table of a system (value: "Executes in Existing Process"), i.e., the lifecycle of the service is different than the lifecycle of the process. An example for this is the *Tomcat* servlet engine of the Apache Jakarta project, which can be invoked in two different ways: It can either be executed as an independent process, or run within the Apache web server.

### 4.2.2 Provide a means for expressing nested Software Features

*CIM_SoftwareFeatureComponent*, an association defined in the CIM Application Schema, models a set of subordinate or independent *CIM_SoftwareFeatures*, which are aggregated to form a higher-level *CIM_SoftwareFeature* under the same *CIM_Product*. It therefore helps to group software components according to the functionality they provide. More specifically, this aggregation allows a *CIM_SoftwareFeature* to be contained within another *CIM_SoftwareFeature* (see Figure 3).

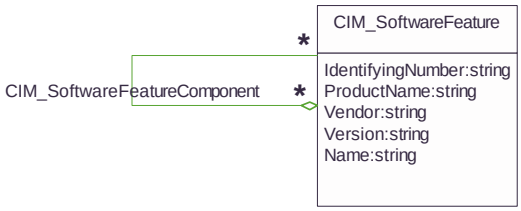


**Fig. 3:** *CIM_SoftwareFeatureComponent*

An example on how this new aggregation can be used is for expressing that "english/french/spanish/german spell checkers" are parts of a "spell checker" *CIM_SoftwareFeature*.

### 4.2.3 Attach additional Semantics to Properties describing their Usage

There is a need to add meta-information to properties of managed objects for classifying them according to how they should be interpreted and used by management systems: Some properties are descriptive, others represent configurable parameters of a resource, yet others contain the status of a managed object. Depending on this qualifier, a management system may, e.g., infer status information of a managed resource by evaluating whether the properties in question are labeled with "state". Another usage could be that a management system forbids the setting of metric parameters (such as counters and gauges) by a system administrator. This is the purpose of the new optional qualifier "PropertyUsage". Its possible values are:

- **Descriptive**: the property contains information that describes the managed element; e.g., vendor, description, caption etc.,
- **Capability**: the property refers to inherent capabilities of a managed element regardless of its configuration, e.g., "VideoController.MaxMemorySupported=128",
- **Configuration**: the property influences or reflects the configurational state of the managed element, e.g., "VideoController.CurrentRefreshRate",
- **State**: the property contains or can be used to derive the current status of the managed element,
- **Metric**: the property is a numerical value representing a statistic or metric reporting performance– and/or accounting–oriented information for the managed element.
- **CurrentContext**: the nature of the property shall be inferred based on the position of its class in the CIM schema. It should be noted that one of the intentions of CIM was to determine for a class whether it represents information related to the aforementioned categories by evaluating whether it has been derived from *CIM_Setting* (for configuration), *CIM_Statistics* (for metrics), *CIM_ManagedSystemElement* (for status), or *CIM_Product* (for capability-related or descriptive information).

### 4.2.4 Distinguishing between multiple Installations of a Product

There are some cases when the same product is present multiple times on a given system, each eventually in a different configuration: A server, for example, may have 2 identical patches for a product (one for the server itself, another for a diskless workstation) installed. A software inventory tool must be able to distinguish between those software features and send back two entries when asked to enumerate the inventory of the server. Another example is the multiple presence of the same software product on a system, such as one "production" and one development version.

One new class and three new associations between the new class and some existing classes of the CIM Application Schema are needed to express such a configuration; they are depicted in Figure 4.

- *CIM_InstalledProduct* is a newly introduced class, which represents the collection of Software Features and Software Elements that make up an installed product. Various properties for distinguishing between multiple instances are defined.
- *CIM_InstalledProductImage* is an association that relates an arbitrary number of *CIM_InstalledProducts* to one *CIM_Product*. *CIM_InstalledProduct* is on the "weak" side of the association because it has to have a *CIM_Product* counterpart for being existent.
- *CIM_CollectedSoftwareFeatures* is an aggregation, which does the same for *CIM_InstalledProduct* what *CIM_ProductSoftwareFeatures* does for *CIM_Product*. It aggregates the various *CIM_SoftwareFeatures* into a *CIM_InstalledProduct* and allows one e.g., to find all the product installations in which a software feature is represented.

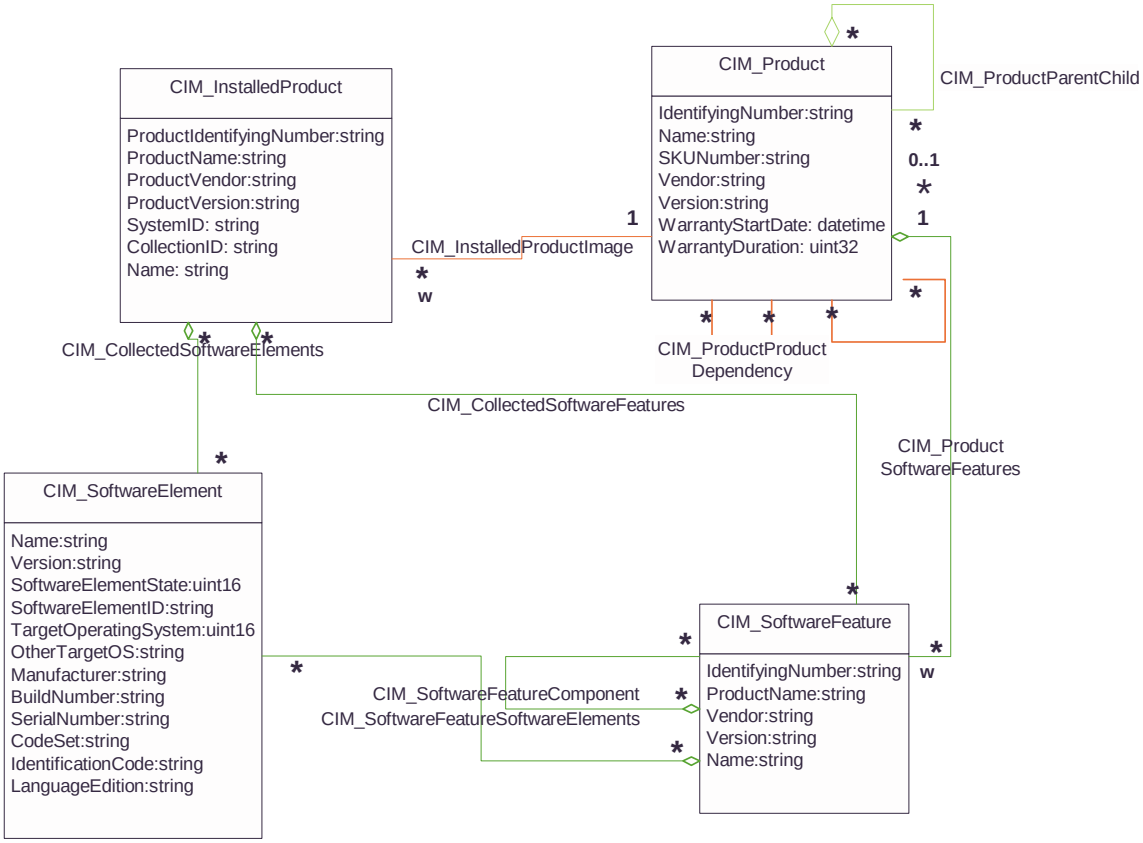


**Fig. 4:** Distinguishing between multiple Installations of a *CIM_InstalledProduct*

- *CIM_CollectedSoftwareElements* is an aggregation that collects various *CIM_SoftwareElements* into a *CIM_InstalledProduct*. This aggregation is used, e.g., to find all the product installations in which a software element is deployed.

### 4.2.5 Add "Stopped" as a possible value of CIM_ManagedSystemElement.Status

One of the most important properties of a managed object is its operational, administrative, or usage status. Such information is captured in the property `Status` of the class *CIM_ManagedSystemElement* in the CIM Core Schema. Typical examples for the operational status of a *CIM_ManagedSystemElement* are: "OK", "Degraded", "Stressed", "Pred Fail" (i.e., high likelihood of a failure). Examples of non-operational status values of a *CIM_ManagedSystemElement* are: "Error", "Non Recover", "Starting", "Stopping", "Service", "No Contact". However, yet there was no value to express that a *CIM_ManagedSystemElement* is not activated. This is the purpose of the newly introduced value "Stopped" whose semantics are as follows: The *CIM_ManagedSystemElement* exists, but is not operational; it has not encountered a failure but has been purposely made non-operational (either by the *CIM_ManagedSystemElement* itself, or by a management system).

### 4.2.6 Provide a consistent mechanism for incremental changes of CIM_Setting

If a single property of a managed object needs to be set to a specific value, this is usually done in CIM by invoking the `SetProperty()` operation (defined in [5]) and specifying the object and property names together with the new property value as parameters. If several properties of one (or more) managed objects need to be set *simultaneously*, CIM provides a mechanism that is based on the object class *CIM_Setting*, defined in the CIM Core Schema (see Figure 5).

As mentioned in section 4.2.3, the state of a base object is carried in (a subclass of) *CIM_ManagedSystemElement* and its associated configuration objects, i.e., instances of *CIM_Configuration*.

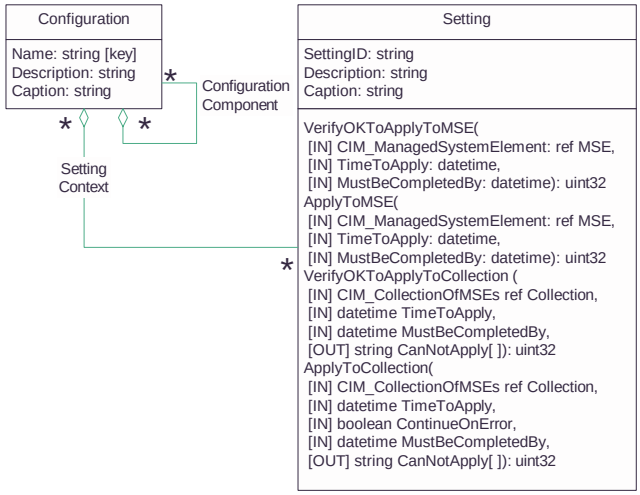


**Fig. 5:** *CIM_Setting* and *CIM_Configuration* (before changes)

The purpose of *CIM_Setting* is to simultaneously ("all or nothing") set parameters of a potential configuration: [4] refers to this as "loosely transactional" behavior (and discusses the procedure in greater detail). This is done by creating an instance of *CIM_Setting*, adding the parameters and their values, verifying whether the parameters can be set at once by invoking the method `verifyOKtoApplyToMSE()` of the *CIM_Setting* object and, after a positive response has been received, jointly setting the parameters on the target object by invoking the `ApplyToMSE()` method of the *CIM_Setting* object. Note that the target object is usually an instance of a subclass of *CIM_ManagedSystemElement*.

When populating the *CIM_Setting* instance with properties and values, it is necessary to provide a reference to every property of the target object. This is the role of the `MODELCORRESPONDENCE` qualifier, which contains the name of the property on a target object for every property to be set. Note, however, that *CIM_Setting* may also be used to hold actual configuration data for which no corresponding property exists in the base object; these properties do not carry the `MODELCORRESPONDENCE` qualifier. Applying such properties may cause "invisible" side effects on the behavior of the base object.

If a setting needs to be applied only to a subset of the object properties (while leaving the other properties of an object at their current value), the `NULLVALUE` qualifier is used for the properties that remain unchanged.

This, however works only for properties of datatype "String" because boolean–, float– or integer–type properties do not have the notion of an "undefined" value. Consequently, there is no consistent way to facilitate *incremental changes of settings*, i.e., to filter which properties of *CIM_Setting* should be changed (by means of invoking `ApplyToMSE()`) and which properties should keep their current value.

Our extensions to *CIM_Setting* address the problem that there was, until now, no way in CIM to indicate an unset value in a consistent manner for boolean, integer and float datatypes. This is necessary for *incrementally* updating a subset of all the properties in an instance of (a subclass of) *CIM_Setting* while leaving the other properties unset or defaulted. Consequently, it prevents the danger of inadvertently resetting a property whose value has been previously set by means of the `MODELCORRESPONDENCE` qualifier. The mechanism works for any datatype and mirrors the existing `VerifyOKToApplyToMSE()` and `ApplyToMSE()` methods and adds an array of strings named `PropertiesToApply` as input parameter. This array contains a list of the property names whose values will be first verified and then applied, thus eliminating the need for a `NULLVALUE` qualifier. The new methods are called `VerifyOKToApplyIncrementalChangeToMSE()` and `ApplyIncrementalChangeToMSE()`, respectively. Since it is possible to change the properties of either a single *CIM_ManagedSystemElement* or a collection of *CIM_ManagedSystemElement*, the same mechanism has been defined for *CIM_CollectionOfMSEs*; thus, *CIM_Setting* carries two additional methods `VerifyOKToApplyIncrementalChangeToCollection()` and `ApplyIncrementalChangeToCollection()`.

# 5 Conclusions and Outlook

We will now summarize in section 5.1 the lessons we learned during our work and give an overview of the current work items of the Application Working Group in section 5.2.

## *5.1 Lessons learned*

We initially started with the hypothesis that our schema extensions would be confined to the Application Schema. However, as section 4.2 demonstrates, the developed extensions required modifications to the Core, System and Application schemas. It turned out that the expressiveness needed for applications must often be defined within other schemas than the Application Schema. This implied the involvement of other working groups in the acceptance process of the proposed schema extensions, conducted by the CIM Technical Committee.

As seen with other management frameworks in the past, one of the most complicated mechanisms relates to uniquely name and identify object instances. The fact that CIM uses keys for identifying object instances and the mechanism of key propagation by means of "weak" associations leads, among other, to the fact that we were unable to reuse the already existing *CIM_Action* and *CIM_Check* object classes for defining generic operations that can be applied to other classes than *CIM_SoftwareElement*. A seperate schema for defining such CIM Operations is currently being developed by the Application Working Group. In addition, some object classes require keys that are composed of several properties (in extreme cases, up to seven properties are needed for uniquely identifying an object instance), while others (such as *CIM_Service*) lack keys to distinguish between two instances. Adding or removing keys, in particular to classes defined in the Core Schema, results in breaking existing CIM implementations and requires a CIM version change. To sum up, the best candidates for keys are ID numbers; scoping objects with keys turned out to be problematic and should be done with associations instead.

The mechanism based on *CIM_Setting* for applying configurations to one or more managed object properties simultaneously also turned out to be fairly complex. The change request described in section 4.2.6 provides a means for consistently applying settings to properties, regardless of their datatype.

Understanding the intricacies of CIM modeling requires a serious effort and commitment; the more than 700 defined object classes and a multitude of association and aggregation relationships make CIM a fairly

complex model. The principle of extending CIM schemas by means of inheritance implies that changes within the Core or the Common Schemas impact all the classes within other schemas that inherit from them. However, it is hard to predict the impact of a change due to the sheer number of classes and inheritance hierarchies whose depth spans sometimes more than seven levels. This problem is exacerbated by the fact that different Common Schemas use the classes defined in the Core Schema in very different ways. The root cause of this issue is that – despite the availability of various documents from the DMTF (listed, among other, in the references of this paper) – the finer semantics and the intent of the existing schemas are often not sufficiently documented, difficult to ascertain and sometimes not agreed upon. A very rudimentary mechanism to find the classes who will be affected by the change in one core class is to search with a text editor for the name of the superclass that is going to be modified, because the MOF files contain the name of the superclass. This, however, yields only the direct subclasses and does not help if the inheritance relationship is indirect (i.e., there are one or many classes on the inheritance path between a class and its subclasses). One of the many functions of the CIM Technical Committee is to discover and help resolve the impact of changes in a superclass. Clearly, it would be desirable to obtain the CIM schemas in a format that can be handled by state-of-the-art CASE tools, which provide appropriate search and reporting capabilities. However, UML-compliant CASE tools will not be able to handle the CIM extensions to UML (e.g., keys, "weak" associations, qualifiers), which were described in section 3.2. Most of these extensions would then have to be entered in the comment field of a UML artifact, which makes them difficult to track with report (and code) generators.

The DMTF releases new schemas containing incremental changes every six months, without tying itself to the state of completion of schema developments. Therefore, the initial extensions published in CIM 2.5 and discussed in this paper set the stage for ongoing work, which is described in the following section.

## *5.2 Ongoing Work*

We will now discuss the ongoing work of the Application Working Group, which aims at providing detailed models for establishing distributed application manageability and builds on the work that has been described in this paper. Manageability instrumentation refers very generally to all the information an application exposes about itself that lends to its being well managed. This includes both static and dynamic information. Static information includes identifying and descriptive information, static policies, application topology, and operations available. Dynamic information includes metric values, current configuration, and current state. Both of these types of information are generally "pulled" by management systems. It is also important that applications be able to "push" events to management systems. With the adoption of an event forwarding mechanism in CIM 2.5, event-driven management can be finally realized in a CIM/WBEM environment.

The types of manageability that applications should instrument for are categorized as follows:

**Identification and Description:**
Description of the application in human readable form as well as technical, installation and deployment information. This might include name, version, operating system, install date, time started, current state, machine name, IP address, port, user id, etc.

**Operations:**
Operations are actions performed by or for an application and are supposed to be defined and activated at runtime. Some well-known operations are start, stop, reset, status, etc. Externally invokable operations are executable files, line mode commands, scripts, etc. Internally invokable operations include methods and functions calls that may make up a well-known application programming interface (API). Operations may (or may not) affect the operational or configuration state of the application.

**Configuration:**
Configuration items are attributes that have a lifespan greater than the execution time of the application instance. Getting/setting configuration attributes typically affects the current state of the application.

**Metrics:**
Metrics are measurable attributes of the application. Metrics can be simple - a single value, like a counter or

they can be complex where they are composed of algorithms including configuration information, system information, and/or other metrics. Getting/setting metrics typically does not affect the current state of the application.

**State:**
Since applications can be composed of other applications and their software features and dependent on relationships to other applications or resources, an application's state may be complex to calculate. State would have to relay what the current state is, its desired state, its last state, and its valid set of states. State changes may propagate up and down the "application food chain".

**Events:**
Events are notifications issued asynchronously from an application to relay information about its current state, a state change, or an unacceptable condition. Applications should issue standard events during their lifecycle, especially cold start, warm start, stop, failure, and refresh.

**Application Topology, Relationships and Dependencies:**
Application topology defines the components of the application and the relationships/dependencies between them. These components and relationships can change during the lifecycle of the application. Dependencies are a special kind of relationship; they describe if, and to which extent, application components require the presence of other components in order to function properly. This information is needed for problem determination in case of faults or performance degradation.

**Monitoring and Thresholding:**
Monitors are the definition of how the values of metrics are to be inspected and tested. Normally they are regular samplings of metric data, which are then evaluated against threshold policies. Based on the evaluation, an event may be issued to notify any interested parties that the metric or condition is acceptable or unacceptable.

**Heartbeats:**
Heartbeats allow the application to demonstrate that it is alive and well. The application regularly announces that it is in a healthy operational state by generating events automatically and repeatedly. The system listening for heartbeats must understand that if the timeout period expires between the heartbeat events, the application may not be functioning correctly.

**Exercisers:**
The Exerciser is a transaction or command invoked from the outside of the application that exercises the application in some fairly complete way to determine if it is really alive and able to deliver its functionality in a timely way. What is exercised is a function of the application itself.

The Applications Working Group is currently extending the existing CIM Schemas to make sure that all of these aspects can be modeled appropriately for a wide range of applications. In particular, the working group has developed first drafts in the area of Operations and Metrics and intends to propose these schemas for adoption in the next version of the schemas, to be released at the end of this year.

## Acknowledgments

The authors would like to thank the members of the CIM Application Working Group for helpful discussions and continuous advice, and their commitment to making this work succeed. Specifically, the authors would like to express their gratitude (in alphabetic order) to Mark Johnson (Tivoli Systems), Takaki Kuroda (Hitachi), Vijay Machiraju (Hewlett Packard), Mike Reynolds (BMC Software), John Sweitzer (Tivoli Systems) and Andrea Westerinen (Cisco Systems).

## References

[1] W. Bumpus, J.W. Sweitzer, P. Thompson, A.R. Westerinen, and R.C. Williams. *Common Information Model: Implementing the Object Model for Enterprise Management*. J. Wiley & Sons, 2000.

[2] Common Information Model (CIM) Version 2.2. Specification, Distributed Management Task Force, June 1999. http://www.dmtf.org/standards/cim_spec_v22/.

[3] Understanding the Application Management Model, Version 1.0. White paper, Distributed Management Task Force, May 1998. http://www.dmtf.org/var/release/Whitepapers/CIM_Applications_wp.pdf.

[4] Common Information Model (CIM) Core Model, Version 2.4. White paper, Distributed Management Task Force, August 2000. http://www.dmtf.org/var/release/Whitepapers/DSP0111.pdf.

[5] Specification for CIM Operations over HTTP, Version 1.0. Specification, Distributed Management Task Force, August 1999. http://www.dmtf.org/download/spec/xmls/CIM_HTTP_Mapping10.php.

[6] Specification for the Representation of CIM in XML Version 2.0. Specification, Distributed Management Task Force, July 1999. http://www.dmtf.org/download/spec/xmls/CIM_XML_Mapping20.php.

[7] XML As a Representation for Management Information - A White Paper Version 1.0. Technical report, Distributed Management Task Force, September 1998. http://www.dmtf.org/standards/xmlw.php.

[8] J. Strassner. *Directory Enabled Networks*. Macmillan Technical Publishing, 1999.

[9] R. Sturm and W. Bumpus. *Foundations of Application Management*. J. Wiley & Sons, 1998.

[10] OMG Unified Modeling Language Specification. Version 1.3 ad/99-06-08, Object Management Group, June 1999.