Common Information Model (CIM) Metrics Model

Version 2.7

June 16, 2003

Abstract

The DMTF Common Information Model (CIM) is a conceptual information model for describing computing and business entities in enterprise and Internet environments. It provides a consistent definition and structure of data, using object-oriented techniques. The CIM Schema establishes a common conceptual framework that describes the managed environment.

The Metrics Model is just one component of a more general set of models for managing runtime applications. The charter for the DMTF's Applications Working Group concerning metrics is the following:

The Application Working Group will define a common model for metric data. A metric is a value that characterizes the state or performance of a resource, and could be any of several data types, including both numeric and non-numeric values. The DMTF's Distributed Application Performance schema will be folded into this more general Metric Model. Other common schemas may define extensions to the basic Metrics Model, defining metrics specific to the schema domain. It is expected that metrics would be used in gets, queries, indications, etc.

This work on metrics was completed initially as part of version 2.2 of CIM. This white paper describes the unit of work concept of version 2.7 of the CIM Metrics Model, which takes the model further.

Notices

DSP# 0141 Status: Preliminary

DMTF is a not-for-profit association of industry members dedicated to promoting enterprise and systems management and interoperability. Members and non-members may reproduce DMTF specifications and documents for uses consistent with this purpose, provided that correct attribution is given. As DMTF specifications may be revised from time to time, the particular version and release date should always be noted.

Implementation of certain elements of this standard or proposed standard may be subject to third party patent rights, including provisional patent rights (herein "patent rights"). DMTF makes no representations to users of the standard as to the existence of such rights, and is not responsible to recognize, disclose, or identify any or all such third party patent right, owners or claimants, nor for any incomplete or inaccurate identification or disclosure of such rights, owners or claimants. DMTF shall have no liability to any party, in any manner or circumstance, under any legal theory whatsoever, for failure to recognize, disclose, or identify any such third party patent rights, or for such party's reliance on the standard or incorporation thereof in its product, protocols or testing procedures. DMTF shall have no liability to any party implementing such standard, whether such implementation is foreseeable or not, nor to any patent owner or claimant, and shall have no liability or responsibility for costs or losses incurred if a standard is withdrawn or modified after publication, and shall be indemnified and held harmless by any party implementing the standard from any and all claims of infringement by a patent owner for such implementations.

For information about patents held by third parties that have notified the DMTF that, in their opinion, such patent may relate to or impact implementations of DMTF standards, visit http://www.dmtf.org/about/policies/disclosures.php.

Table of Contents

Table of Figures

1. Introduction

This white paper describes the CIM Metrics Model, a set of classes to support gathering and managing dynamic metric information. Effective with the current CIM release, this work includes both a metric to gather the Unit of Work information (Unit of Work metric), and another to work with single valued metrics (called Base Metrics). Previously, through version 2.6 of CIM, the Metrics Model consisted simply of the Unit of Work Model. Now that the Model has been extended in CIM 2.7, the term UoW Model is used to discriminate the Units of Work concept from the other parts of the Metrics Model, which are labeled Base Metrics. UoW comprises all classes that relate to the Units of Work concept.

1.1 Overview

The Unit of Work Concept - A prime goal of administrators is to ensure availability and timely response of the application systems. In the first place, statistics, aggregated metrics, as well as indications are used as indicators for performance problems and system failures. The continuous observation and comparison against thresholds of these statistics, metrics, and indications is what application monitoring is about. In order to calculate these statistics, “raw” measurement data is needed. Such data is ideally produced by the instrumentation of the application and describes an instance of an individual operation or software "action" executed by the application system. The operation or action may address the entire response to a user’s request or part of the response, such as the "important" part of a database access that is required to fulfill the user’s request, or a combination of both. Every time the request comes in, the instrumentation determines the measurements for which it was built - resulting in the calculation of a metric and/or some measurement of a unit of work.[1]

The values that are measured may comprise the duration of the operation or action, the amount of memory used, the business process the unit of work belongs to, etc. Once the unit of work has been measured, the data can be used either internally for statistics and aggregated metrics calculation or is surfaced by means of an agent to some management application that regularly collects the data and performs additional calculations.

Getting unit of work data and using it as a basis for monitoring is only the first half of the story. Detecting a performance bottleneck does not mean that the cause is known. The analysis of the problem starts with the detection. So, additional business process-specific statistics may be needed to get closer to the source of the problem, since pre-defining statistics for all possible views of a complex system is very difficult. Analysis statistics can only be provided if the unit of work data is available for calculation. It is likely that the activity of individual business actions must be investigated in order to get down to the cause of a failure or bottleneck. Therefore, the measurement data of an individual unit of work is needed. The analysis problem

[1] Note that the term ‚unit of work’ can be defined arbitrarily and does not need to correspond to a transaction.

is even more complex if the business action spans over several services or systems, and thus a net of related unit of works makes up the business action.

The principal goals of the UnitOfWork Metrics Model are as follows:

1. Define a model for representing UnitOfWork metrics and their definitions; an instance of a metric should exist only when a definition of its characteristics is present. Stated differently, a metric exists only within the scope of its definition. This allows, among others, the enumeration of all the metric instances for a given metric definition.

2. Provide a mechanism for dynamically (i.e., at runtime) associating both metrics and their definitions with a logical element.

The Base Metric Concept - The base metrics concept generalizes the previously introduced UoW concept. It is not only focused on raw data related to response times of requests or actions, but allows modeling any type of raw and aggregated metrics. It also adheres to the separation of meta data (definitions) and the values, which alleviates all cases where late bound data retrieval mechanisms are needed.

2. The Metrics Model

2.1 Background and Assumptions

The Metrics Model began with the Unit of Work metric. The UoW Model was originally within the DAP (Distributed Application Performance) Working Group (later renamed the Metrics Working Group). The focus of DAP was specifically to develop a CIM Schema for measurement information about units of work, such as would be measured with Application Response Measurement (ARM) tools and standards. The original scope was oriented around modeling the concept of the **UnitOfWork**.

The UoW Model today includes the classes and associations to model the concept of UnitOfWork, the definition of UnitOfWork, the creation of instances of UnitOfWork and the metrics associated with a UnitOfWork.

In 2000, the DMTF Applications Working Group began to develop a model specifically for the management of runtime applications (the initial applications model deals primarily with software distribution). One of the specific objectives of that work was the modeling of metrics, such as counters and gauges. It became apparent that the UoW Model of the Metrics group and the work of the Applications Working Group on metrics had common objectives so the groups were merged so that the current and future work of the Metrics Model is expected to be part of the Applications Working Group.

2.2 Conceptual Areas Addressed by the Model

What are the major issues that are tackled by the CIM Metrics Model?

1. Addressing performance, availability, and fault management.
2. Delivering flexible, dynamically extensible meta data with very fine granularity.
3. Identifying (or associating to) individual operations, software actions, or generic managed resources – like resources of an operating system.
4. Publishing the semantics of the measurement data, i.e., providing a definition for the management application in order to understand what the data means.
5. Relating the measurements to the entity (application system, service, component, etc.) that executed the measured operation or action.

For the UoW model, the following issues are also important:

6. Showing the currently pending or recently finished actions.
7. Relating the measurements hierarchically (granularity) as well as sequentially (correlation).

The measurement data that is to be described with the Metrics Model is not necessarily provided by application instrumentation (i.e. intrusively), but instrumentation is the more powerful means to provide the right semantics needed for fault and performance management. Non-intrusive measurements are applicable if existing, non-instrumented applications need to be considered.

CIM Metrics may be based on top of existing performance measurement infrastructure, so that no data is replicated. In this case, CIM is only used as a standardized interface to existing performance gatherers. As CIM is an open standard, in the future customers may combine best-of-breed performance tools for various needs (like storage management, operating system, or middleware applications) from different vendors and feed the (correlated) data into other systems management applications of their choice.

2.3 Understanding Base Metrics Model

CIM users often desire metric objects that model designers have not provided. Rather than fill more and more CIM Schema with standardized objects, the Metrics Model supports externally defined metrics, which add dynamic properties to existing classes.

2.3.1 Differences from the CIM Statistics classes

There are several differences between CIM Metrics and CIM Statistics classes (CIM_StatisticalInformation or its newer equivalent, CIM_StatisticalData):

- Metrics always contain exactly one value with meta data associated to a managed element, so metrics generally have a very fine granularity. Statistics classes usually contain several properties. If the exploiter applications need a given set of metrics together, and if this set can be identified beforehand for all relevant exploiter applications, statistics have the advantage that they are faster, because they transfer several metrics in one object instance.

- As a consequence of the fine granularity, it is possible to add meta data, which precisely refers to only one property.

- Applications can reference to single metrics using associations, instead of cumulated classes containing several metrics, which may not all be relevant for the association.

- It is assumed that metrics define a variety of analyses (such as time series analysis) on statistical or other numeric properties of CIM. So, "standard" statistics are defined in subclasses of CIM Statistics or StatisticalData, while these higher-order analyses are defined in metricse.

- Late binding-like concepts can be realized. As an example, it would be possible to have a systems management application consuming data from a performance monitor for storage-related metrics. The performance monitor could be exchanged with a different performance monitor from another vendor without the need of change in the exploiter application.

2.3.2 Base Metric Class diagram

Base Metric Definition instances and related Base Metric Value instances can be associated to generic instances of subclasses of CIM_ManagedElement. By traversing the associations between the metric instances and existing CIM resources, it is possible to obtain all metric values associated to a given resource instance and related to a given metric definition.

The class diagram looks like this:

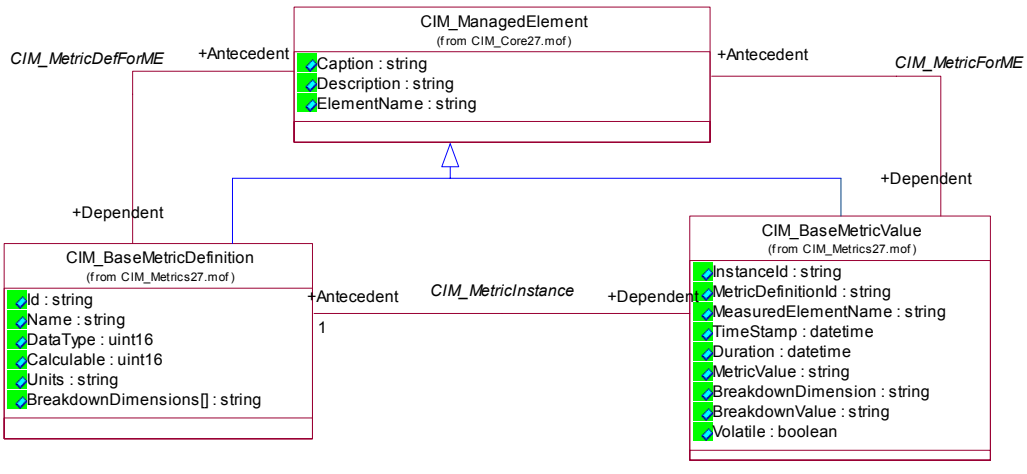

Figure 1: Base Metric Definition/Base Metric as Subclasses of CIM_ManagedElement

In the following, all important classes of the picture above are described.

Class CIM_BaseMetricDefinition

A CIM_BaseMetricDefinition instance represents the definitions aspects of a metric, i.e., a definition template containing meta data about a new metric. Since CIM_BaseMetricDefinition does not capture metric instance information, it does not contain the value of the metric. The associated class CIM_BaseMetricValue holds the metric value. The purpose of CIM_BaseMetricDefinition is to provide a convenient mechanism for introducing a new metric definition at runtime and capturing its instance values in a separate class. Vendors of managed resources may make use of this mechanism if a small subset of a potentially large and heterogeneous amount of performance-related data needs to be exposed. Additional meta data for a metric can be provided by sub classing from CIM_BaseMetricDefinition. The CIM_BaseMetricDefinition should be associated with the CIM_ManagedElement(s) to which it is applied.

Properties:

- *Id*: key; OSF UUID recommended
  This is an opaque key and has no special semantic meaning. The usage scenario should be that the CIM client application may ask for all CIM_BaseMetricDefinitions associated to a given resource.

- *Name*: descriptive name of the metric (like metric "RequestRate").

- *DataType*: standard CIM data type like string, sint32, ...

- *Calculable*: Non-calculable or Summable or Non-Summable.

– "Non-calculable": a string. Arithmetic makes no sense.
– "Non-summable": it does not make sense to sum this value over many instances of BaseMetricValue. An example would be a metric that measures the queue length when a job arrives at a server. If the average queue length when each job arrives is 33, it does not make sense that the average queue length for 100 jobs is 3300. It does make sense to say that the mean is 33.
– "Summable": It is reasonable to sum this value over many instances, such as the number of errors.

- *Units*: identifies the specific units of a value, like Bytes or Packets.

- *BreakdownDimensions*[] (array of strings): Defines one or more strings that can be used to refine (break down) queries against the BaseMetricValues along a certain dimension. An example for a dimension is a transaction name, allowing a break down of the total value for all transactions into a set of values, one for each transaction name. Other examples would be application system name, or user group name. The strings are free format and should be meaningful to the end users of the metric data. The strings indicate which break down dimensions are supported for this metric definition by the underlying instrumentation. It can then filter out the metric it would like to use by looking at the Name property.

Class CIM_BaseMetricValue

Each instance of CIM_BaseMetricValue represents a metric value.

Properties:

- *InstanceId*: key property.
- *MetricDefinitionId*: foreign key for *CIM_BaseMetricDefinition.Id*
- *MeasuredElementName*: descriptive name for the managed element being measured.
- *MetricValue*: the measured value itself.
- *TimeStamp, Duration*: Time range to which the metric value applies. *TimeStamp* identifies the time when the value of a metric instance is computed. Note that this is different from the time when the instance is created. For a given CIM_BaseMetricValue instance, the *TimeStamp* changes whenever a new measurement snapshot is taken if *Volatile* is true. A management application may establish a time series of metric data by retrieving the instances of CIM_BaseMetricValue and sorting them according to their TimeStamp.

  Duration represents the time duration over which this metric value is valid. This property should not exist for time stamps that apply only to a point in time but should be defined for values that are considered valid for a certain time period (e.g. sampling). If the "Duration" property exists and is not *null*, the TimeStamp is to be considered the end of the interval.

2.3.3 Association classes

The associations are central to the Base Metrics Model.

To find out which metrics are available for a given resource, one has to traverse the *CIM_MetricDefForME* association between *CIM_ManagedElement* and *CIM_BaseMetricDefinition*.

Traversing from *CIM_BaseMetricDefinition* to *CIM_BaseMetricValue* can result in the set of all metric values for a given base metric definition.

Finally, the *CIM_MetricForME* association can help find all metric values for a given *CIM_ManagedElement* resource class.

2.3.4 Combining providers of different vendors

In enterprise server environments, it is expected to be common to have various CIM metrics providers active at the same time. For example, there may be a database monitor, a self-tuning systems management application with some externalized status information, and a SAN monitoring application.

In CIM, each class has exactly one provider. Therefore, it is recommended that every CIM metrics provider subclasses from *CIM_BaseMetric* and *CIM_BaseMetricDefinition* (like *IBMDB_BaseMetric* and

IBMDB_BaseMetricDefinition). If an exploiter application would like to ask all metrics providers, this is still possible due to the inheritance mechanisms implemented in the CIMOM.

2.3.5 Breakdown dimensions

Some performance monitors allow break down of the value of some metrics along a breakdown dimension, such as a process or a service class. This effectively adds another dimension to the performance metric: you cannot only see the *RequestRate* of a disk drive as a whole, but also the *RequestRate* initiated by a given process. It is then easy see which process is the most active user of that disk resource.

2.4 Understanding the UoW Model

The unit of work concept stems from the original DAP Model whose objective was to define a common model for probing the response time of distributed applications in heterogeneous environments, at runtime. The unit of work measures time for some work to be performed, and can attach other metrics to provide additional information.

Originally defined for transaction response time measurement, the Unit of Work concept can be extended to address a variety of runtime entities, such as:

- Batch jobs
- User-initiated interactive operations
- Transactions executed under the control of a TP Monitor
- Short server transactions, such as a database read
- Round trip network delays

The term unit of work is used instead of “transaction” because the latter term may imply specific behavior and, as the list above shows, there are many possible units of work beyond mere transaction processing.

There are two primary properties to any unit of work measurement:

- Time it took to complete the unit of work (or elapsed time if still executing)
- Status of the unit of work: Active, Suspended, Completed (with unknown state), Completed Good, Completed Failed, Completed Aborted

In addition, it is very useful to understand if a unit of work depends on another unit of work, such as one invoking a second unit of work, waiting for data to be returned, and then continuing processing. This relationship between a unit of work and its sub-units may occur repeatedly and at multiple levels. The interplay and outcomes of the units of work affect metrics and status. For example, it is common that a failure in a

sub-unit will cause its calling unit of work to fail, and this failure may then propagate up the calling chain.

The central class of the UoW Model is CIM_UnitOfWork. It represents an individual software action or operation – a unit of work. It has an identity (which is its key) and provides contextual information (e.g. the UserName or MutualContextId) and, most importantly, the response time measurement values along with the execution status of the instance. Each instance needs to reference additional information that allows a management application to understand the semantics of the unit of work. Therefore, CIM_UnitOfWork instances are associated via CIM_StartedUoW to their corresponding definition (CIM_UnitOfWorkDefinition). The unit of work instance can also be related to an element (i.e. some instance of CIM_System, CIM_Service, etc.) that executes/executed the unit of work via the association, CIM_LogicalElementPerformsUoW.

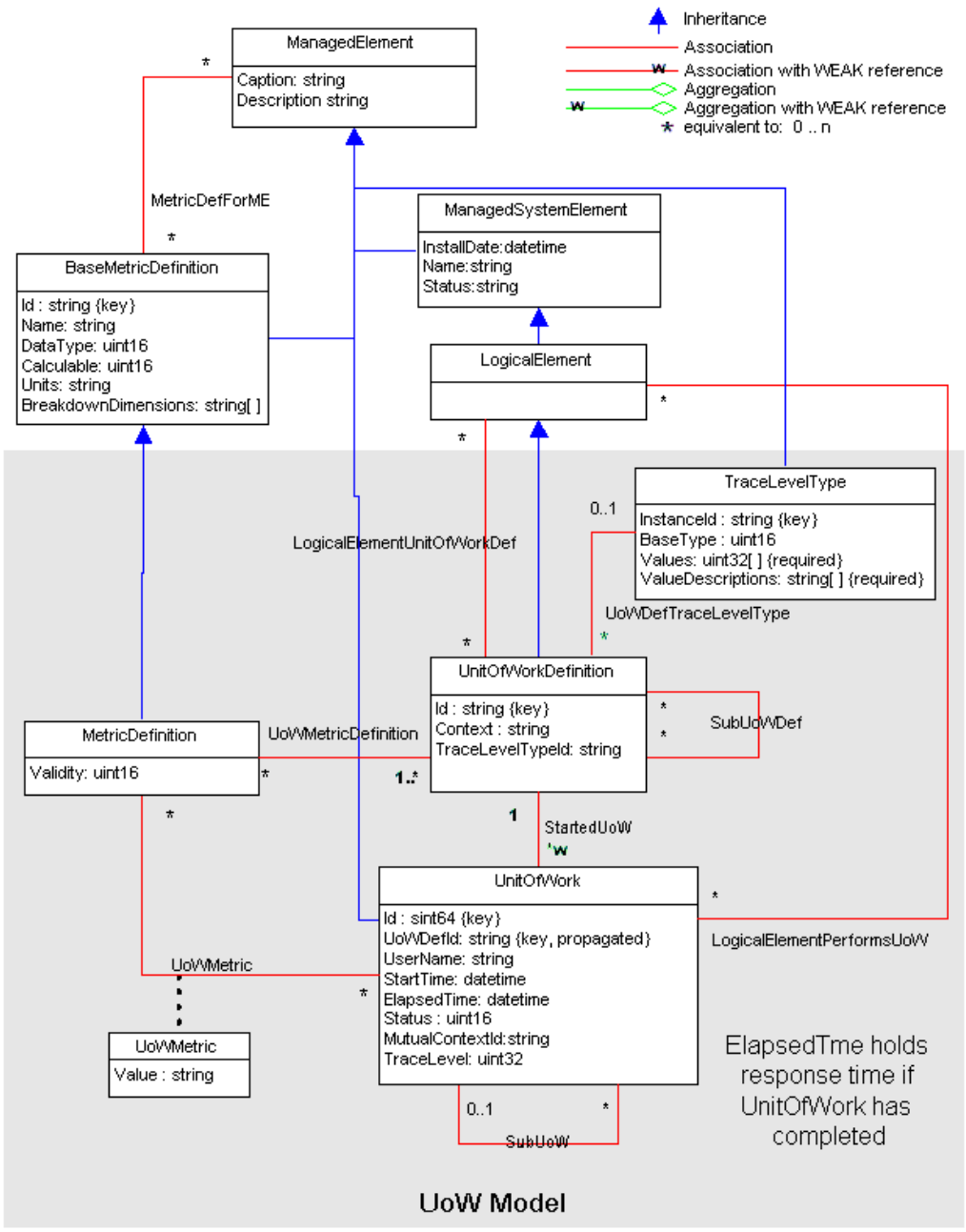

Figure 2: UoW Model – Metrics Model Version 2.7

An important feature of the model is the association CIM_SubUoW. The use of the association allows correlation of instances of CIM_UnitOfWork. The correlation may express several situations:

- Hierarchy: A high-level unit of work can be broken down to several smaller units of work, i.e., the granularity of measurements is refined. Note, however, that accumulating an overall response time from these smaller units of work is often a complex and error-prone activity. Sub-transactions may execute

serially or in parallel and there are often small time components that are not measured. Thus, ten sub-transactions running in parallel and all taking one second to complete only delay the parent transaction one second, not ten seconds. There has been discussion with the ARM community that additional semantics be defined to indicate if a sub-transaction was blocking or non-blocking, but this has not moved beyond the stage of discussion to date.

- Selective hierarchy: The execution of a high-level unit of work may be refined by its "more important" fine-granularity sub-units and measurements. Database access or communication to services outside the application during the execution of a business action are examples of such selected and fine-grained units of work.

Note that the semantics of CIM_SubUoW is not intended to model sequential relationship of units of work, but for blocking and non-blocking hierarchical calls.

If the unit of work hierarchies are predefined and known before the execution of the action or operation, CIM_SubUoWDef can be used to express the anticipated relationships between the instances. A management application could use such information to determine incorrect execution paths or to prepare graphical representation of the results of the unit of work measurements. Note that an instance of unit of work can only be associated to one parent. This is true because the sub-unit can only be executing in the context of one parent. On the CIM_SubUoWDef side, this is not true – since a definition can be used as a template in many higher level definitions.

When using CIM_SubUoWDef and CIM_SubUoW for correlation, an important issue is cross-namespace associations. Cross-namespace associations come up if the entities that are associated reside in different namespaces (e.g. on different machines), i.e., the instances of CIM_UnitOfWork are provided separately and so would need to be associated by a cross-namespace association. Current CIMOM implementations cannot resolve an association that possesses a reference that points to another namespace (neither local nor remote). This general problem is not the scope of the model. The possible solutions are:

- Organizing the providers in such a way that they operate against one namespace

- Using a correlation information class that implicitly carries the necessary correlation properties (see the extended use case example in Section 4 for details). Some central manager would access the correlation information and could provide the associations when needed.

As described, the correlation context of CIM_UnitOfWork instances is provided by CIM_SubUoW association instances. Thus, the mutual context of CIM_UnitOfWork instances has to be retrieved by following the associations between these instances (associator queries). Still, the mutual context is not named explicitly. Naming the context may be interesting for two situations:

- Displaying all CIM_UnitOfWork instances that participate in an action as a whole (e.g. a table where each mutual context is a row and each participating CIM_UnitOfWork instance is a column)

- Direct retrieval of CIM_UnitOfWork instances that participate in the same context by using one query expression.

CIM_UnitOfWork.MutualContextId names such mutual context of CIM_UnitOfWork instances. It should not replace CIM_SubUoW association instances. It should support better recognition of distributed actions. Note that it is up to the implementation of the model to generate and distribute the ID in order to ensure that all participating CIM_UnitOfWork instances are consistently assigned the proper mutual context ID.

2.4.1 UoW with Metrics

CIM_UnitOfWork can be extended with metrics. The metrics may have any semantic and data type. Type and semantic are defined in CIM_MetricDefinition. In contrast to version 2.6 of the Metrics Model, CIM_MetricDefinition inherits from CIM_BaseMetricDefinition, which was newly introduced to version 2.7 in the effort to generalize the metrics concept. The metric definition is always defined in conjunction with at least one definition of a unit of work (CIM_UoWMetricDefinition). It is not reasonable to define metrics that are not used.

Like the unit of work definition, the metric definition provides additional information to a management application to understand the semantics and usage of the metric. The value of a metric is maintained in a property in the CIM_UoWMetric association. Providing the metric value by means of this association avoids cluttering the model. The alternative was to model the value of the metric as an additional CIM metric class. This additional class would still need the association to CIM_MetricDefinition and an additional association to CIM_UnitOfWork. This would result in many instances just to obtain a value.

The definition is kept separate from the values of unit of work and metrics because:

1. The model provides great flexibility due to the general semantics of the classes. Therefore, the class definitions need not be changed in order to provide new types of units of work. One needs only to create new instances. Since there may be an infinite number of metrics, this seemed a preferable modeling approach. Note that where "standard" metrics exist, subclasses of these "basic" metric classes may still be defined.

2. Instances of CIM_MetricDefinition can be reused across many different CIM_UnitOfWorkDefinition instances.

3. An alternative approach would be to define qualifiers of the unit of work class. This approach has the same shortcomings of #1, above, and a few additional ones. For example, more than one qualifier would be needed to lay down the definition. And, no container is provided that shows the inherent relationship of these qualifiers.

4. Another alternative would be to combine the properties of the unit of work with its definition. This results in a duplication of the definitional information in every instance, and therefore, model clutter and lack of reuse.

CIM_UnitOfWork.Status and CIM_UnitOfWork.ElapsedTime may be indicators of faulty system behavior or performance bottlenecks. But they are usually insufficient to determine the cause of the fault or performance bottleneck. More detailed information is needed. Such information is often provided by traces that are written during the execution of the unit of work. The administrator needs to know whether traces for some particular unit of work instance are available. CIM_UnitOfWork.TraceLevel is the indicator to the admin about the fact that the unit of work instance has traces. The trace level also defines the granularity of the traces produced. It does not indicate where the traces are found and how they are accessed.

The semantics of the trace level are usually implementation dependent. Therefore, the management application may also need to discriminate different semantics (the trace level type: CIM_TraceLevelType) as well as the actual encoding of the applied types (i.e., whether the level is represented by a bit map or just plain numbers and what the different possible levels mean). Note that the association CIM_UoWDefTraceLevelType is not attached to CIM_UnitOfWork since it seemed unacceptable to burden the potentially numerous CIM_UnitOfWork objects with additional associations (as well as to prevent application code behind CIM_UnitOfWork instances of the same definition to implement different trace level encodings). In order to facilitate the detection of semantic difference of trace levels, the TraceLevelType.InstanceId is replicated to CIM_UnitOfWorkDefinition.TraceLevelTypeId. So, the management application is not forced to traverse the association to the CIM_TraceLevelType in order to determine such difference.

The following class descriptions are for the purpose of completeness in the document. They briefly summarize the key points of the classes or add details not listed above description nor stated in the MOF. Further class details can be found in the corresponding MOF file.

Class CIM_UnitOfWorkDefinition

Derived from CIM_LogicalElement, the CIM_UnitOfWorkDefinition class describes units of work (e.g. "transactions") associated with a logical element. This class represents the definitional components of a unit of work, and not the unit itself. CIM_UnitOfWorkDefinition.Id is the key. A 16-byte value (OSF UUID is recommended) that uniquely identifies this definition is expected.

NOTE: Today, UnitOfWork derives from logical element for historical reasons. One of the open questions for future versions is the possibility of making it more general by deriving it from CIM_ManagedElement in the future

Class CIM_UnitOfWork

Each instance of CIM_UnitOfWork identifies a transaction that is either in-process or already completed. Because instances of 'in-process' CIM_UnitOfWork can be very short-lived and there can be a great number of active instances, use of this class as an instrumentation source for determining response time may be incorrect or inefficient, unless the rate and duration of the UnitsOfWork are known. The intended use is to respond to queries about currently active or recently completed units of work. The length of time that a CIM_UnitOfWork instance exists after the unit of

work completes is not defined and should be assumed to be implementation-dependent. This class is weak to its definition CIM_UnitOfWorkDefintion.

NOTE: It must be remembered that Description is a property of CIM_Managed Element and therefore applies to this class. It is recommended that this property not be used to reduce the size of the UnitOfWork class. Both this and Caption can be left NULL.

The keys for CIM_UnitOfWork are:

- *ID* - This is a 64-bit signed integer that uniquely identifies the instance of the unit of work within the context of an instance of CIM_UnitOfWorkDefinition. The use of standard identifiers such as the OSF UUID/GUIDs is recommended.
- *UoWDefId* – Propagated from the UnitOfWorkDefinition.

Some other properties of the CIM_UnitOfWork are:

- *UserName* – Representing the name of the user who started the UnitOfWork, either a real login name or a logical name representing an application.
- *StartTime* – Time the UnitOfWork Started.
- *ElapsedTime* – Upon completion this property contains the total time that it took the unit of work to complete. During execution, it represents the time that the transaction has been executing.
- *Status* - An enumeration identifying the status of the UnitOfWork:

Status	Value	Additional Description
Active	1	
Suspended	2	
Completed	3	Should be used to represent a 'completed' transaction whose status ('good', 'failed' or 'aborted') is unknown.
Completed Good	4	
Completed Failed	5	
Completed Aborted	6	Should be used when a UnitOfWork has completed but was not allowed to end normally. An example would be when the Stop or Back buttons are selected from a web browser, before a page is fully loaded.

Class CIM_MetricDefinition

The CIM_MetricDefinition class defines a metric that could be associated with a CIM_UnitOfWork. It describes the metric value and behavior (meta data for the metric).These metrics usually describe some aspect of a CIM_UnitOfWork such as how much work was done, or the size of the CIM_UnitOfWork. For example, the size of a print job or the number of pages printed could be metrics of a 'print' unit of work. Almost all properties are inherited from CIM_BaseMetricDefinition.

The properties for MetricDefinition are:

- Validity - The property Validity is an enumerated value describing when the Metric may be considered valid. Some metrics are valid only:

  — at the beginning of a transaction (e.g., bytes to print),

  —while the transaction is running (e.g., percent complete),

  —when the transaction is finished (e.g., pages printed).

  If a metric is valid at more than one of the enumerated values, such as both when the unit of work starts and when it stops, it is recommended to not use Validity, because it is defined as being single-valued (i.e., it is not an array of possible values). The allowable values are: "atStart","inMiddle","atStop".

CIM_UnitOfWork.TraceLevel, CIM_UnitOfWorkDefinition.TraceLevelId, CIM_TraceLevelType, CIM_UoWDefTraceLevelType

The following addresses questions on why the trace levels have been modeled as is:

- Why not make CIM_UnitOfWork.TraceLevel and CIM_TraceLevelType.Values[] of type string instead of uint32?

  It is unusual to use strings to define trace levels, in general, numeric values are used. Therefore, computation is straightforward with uint32.

- Why have CIM_UnitOfWorkDefinition.TraceLevelTypeId?

  In order to increase access efficiency. After looking at the trace level itself, a user needs to know whether two identical trace levels have the same meaning. So, the management application would need to query for the corresponding TraceLevelTypeIds traverses the association between CIM_UnitOfWork and CIM_UnitOfWorkDefinition, and then between CIM_UnitOfWorkDefinition and CIM_TraceLevelType. This is two hops. Placing TraceLevelTypeId in CIM_UnitOfWorkDefinition requires only one hop to retrieve the information that trace levels are to be handled differently.

- Why have CIM_UoWDefTraceLevelType?

  Since TraceLevelTypeId is already present in CIM_UnitOfWorkDefinition (as a foreign key), direct access to the corresponding TraceLevelType instance is guaranteed. In general, CIM requires that classes and their instances should be related by associations and their instances, respectively. Creation of so-called data islands (instances or entire sub-models that are not associated to the rest of the model) is not desired, since management

applications could not navigate through such a model. So, removing CIM_UoWDefTraceLevelType is not an option since CIM_TraceLevelType created a data island. Eliminating CIM_UoWDefTraceLevelType and determining another class to which CIM_TraceLevelType could be attached to is beyond the scope of the current extension. So, for the time being, CIM_UoWDefTraceLevelType is not wrong, supports the standard access to CIM_TraceLevelType instances (query: associators of) and avoids data islands.

Association class CIM_UoWMetric

The association between CIM_MetricDefinition and CIM_UnitOfWork has the additional property Value. Value holds the current value of the metric defined by the referenced metric definition. The value is only meaningful if CIM_MetricDefinition.Validity is interpreted correctly. The value is expected to be encoded as human readable string and not as an array of bytes (e.g. the number 16 (unit32) is encoded as Value =”16” and not as Value = {0x00, 0x00, 0x00, 0x10, \0} or any other encoding).

All other associations have already been described in previous paragraphs.

3. Relationships to Other Standards and Specifications

3.1 Relationship of CIM Metrics to the ARM Specification

The ARM (Applicatison Response Measurement) standard developed as a way to instrument applications for measurement data about transactions response times from the client point-of-view. The CIM UoW Model objective is the definition of the information objects that will represent units of work, a generalization of the initial problem of transaction response time measurement. ARM and the UoW Model are grounded in the same fundamental concept, so it is not surprising that the UoW Model can be used to represent data measured with ARM. While ARM and the UoW Model were developed as components of a common solution to the problem of capturing unit of work information, users are also free to use any other appropriate instrumentation to populate the CIM UoW Model. The following two sections provide some brief background about ARM and a mapping between ARM and the CIM UoW Model.

3.2 What is ARM?

ARM was introduced in 1996 and arose from the necessity to define an instrumentation standard that addressed response time measurements of applications. ARM 1.0 and 2.0 described C APIs, and ARM 3.0 describes Java interfaces that are to be implemented in an ARM agent. These APIs can be used to capture identification, time between initiation and completion, and other metrics information. They can be used to capture performance information for transactions or any

programming function where the time-to-complete is important. The decision to only define APIs is due to the fact that some abstraction from the underlying implementation was needed. The interfaces' methods serve as the communication between the measurement agent and the application. The ARM standard does not define the characteristics of the resulting data, standards for communicating this data to a management system, or APIs for the management system to work with ARM information. The goal was simply to define the APIs that an application could use to capture the information. Conceptually, these interfaces are very simple (like start(), update() and stop(), with a few other supporting APIs).

In version 3.0, ARM interfaces address two different situations:

- The ARM agent is triggered by the application to do the measurements (interface ArmTransaction)
- The ARM agent receives measurements already completed by the application (ArmTranReport)

Both interfaces imply the same data structures to describe the response time, status of the action, identity of the measured action, correlation data, and contextual data. ARM's concept of correlation is that the predecessor (described by a unit of work token called the "parent correlator") of the current action is stored with measurement data.

ARM 3.0 also allows enhancing measurements about a transaction (and its inherent response time metric) with up to seven additional metrics. Therefore, the interfaces ArmTranReportWithMetrics and ArmTransactionWithMetrics are defined to extend the respective transaction interfaces. The metrics fall into four pre-defined categories (gauges, counters, non-calculable numeric values, and strings) and are described by their tailored interface.

ARM requires the application to define the transaction and the metrics prior to the first measurements received or executed by the agent. Thus, additional interfaces for the definition of metrics and transaction types are specified (ArmTranDefinition, ArmMetricDefinition).

3.3 Differences and Equivalences

The most obvious difference between ARM and the CIM Metrics Model is that ARM defines an API that consists of interface methods, used in a pre-defined order. In contrast, CIM_UnitOfWork and its associated classes define a data model that is an abstraction of the metrics themselves and the actual instrumentation. Although the ARM spec also provides diagrams that describe the data model that is implemented by the API, the model has no formal data representation.

The following table lists the CIM classes and the approximate ARM equivalents:

UoW (CIM Metrics 2.7)	ARM 3.0
CIM_UnitOfWork	ARMTransaction ARMTransactionWithMetrics

	ARMTranReport ARMTranReportWithMetrics
CIM_UoWDefinition	ARMTranDefinition
CIM_MetricDefinition	ARMMetricDefinition
CIM User Model	ARMUserDefinition
CIM_UoWMetric	ARMMetric ARMTranReportWithMetrics.MetricValues
CIM_LogicalElementPerformsUoW	ARMSystem, ARMSystemId
CIM_SubUoW	ARMCorrelator
CIM_LogicalElementUnitOfWorkDef	No equivalent
CIM_TraceLevelType	No equivalent

There is no semantic gap between the ARM API and CIM, though clearly the ARM classes are a particular implementation of the more abstract CIM Schema. The following table compares the Metrics Model with ARM 3.0 at a more detailed level:

UoW (CIM Metrics 2.7)	**ARM 3.0**
CIM_UnitOfWork	ARMTransaction, ...
• StartTime • ElapsedTime • CIM_SubUoW.Antecedent • CIM_SubUoW.Dependent • Id • MutualContextId • TraceLevel	• StopTime • ResponseTime • ParentCorrelator • CurrentCorrelator • TransactionHandle • No equivalent • CurrentCorrelator.TraceFlag
CIM_UnitOfWorkDefiniton	ARMTranDefinition
• Context • TraceLevelTypeId	• ApplicationName • No equivalent
CIM_MetricDefinition	ARMMetricDefinition
• Calculable	• Subclasses

• Units • Datatype • Validity	• No equivalent • Subclasses • ARMTransactionWithMetrics.SetMetricValid()

Although, the CIM_UnitOfWork concepts matches ARM very well, some conceptual differences can be identified. The most important difference addresses the metrics. In contrast to ARM, CIM introduces a concept of metrics that is more flexible and more general.

UoW (CIM Metrics 2.7)	ARM 3.0
Model	API
• No data access interface provided • Abstracted from implementation	• Data access defined by methods • Implementation issues (factories)
No pre-defined metrics types (values=strings)	Pre-defined metric types (10 types)
Number of metrics not limited	Max of seven metrics

4. Examples of the UoW Model

4.1 A Holistic Example of the Use of UoW Model

In order to demonstrate the usage of the UoW Model, a very simple example has been chosen. The example uses all features of the model.

Assume a simple application system that browses a catalog (the Catalog Application System). The catalog is stored in a database. The only function provided by the system is searching the catalog. When a catalog search request is issued against the system, the request is validated (format of the query) and handed to a component (DB SAP = database service access point) that executes the DB query. Finally, the results of the query are prepared for presentation and sent back to the requestor. The system is intelligent enough to work on several requests in parallel.

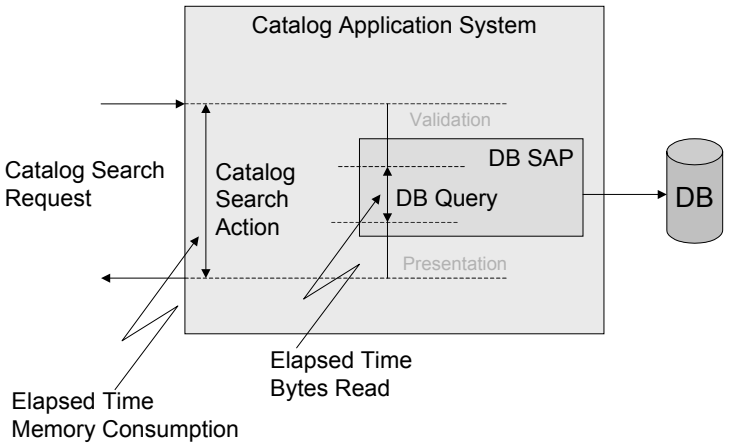

Figure 3: Example Scenario

The developers need information on blocking requests, as well as detailed performance information of the requests to optimally tune their catalog system and database according to specific user needs. They want to measure the entire search, including duration (elapsed time) and its memory consumption. Since the database access is crucial they decide to also measure the DB query in terms of duration and the bytes that have been received (read) from the database during each call.

4.1.1 Unit of Work Definitions and Metric Definitions

For the sake of simplicity, the classes of the Metrics Model have been used as is. It is up to the designer to decide whether derivation is necessary or not. But, the derivation for CIM_ServiceAccessPoint is necessary since this class is defined as abstract, and so cannot be instantiated. The catalog system is represented by an instance of CIM_ApplicationSystem (CatalogSearchSystem). The database access component (DB SAP) is modeled as MY_DBSAP:CIM_ServiceAccessPoint (DBAccess). The Hosted-relationship is not depicted, but DBAccess is assumed to be hosted by CatalogSearchSystem. The application may be described by different instances of CIM_LogicalElement.

As mentioned earlier, the system activity comprises two actions: the catalog search and the database query, whereas the query is always executed within the scope of the catalog search (nested unit of work). These two action types are modeled as different instances of CIM_UnitOfWorkDefinition. Their nesting relationship is expressed by an instance of CIM_SubUoWDef. Instances of CIM_LogicalElementUnitOfWorkDef assign the definitions to the entity that actually executes the units of work. In our case, this is CatalogSearchSystem and DBAccess. In principle, the DBQuery definition could also be attached to CatalogSearchSystem, since the system includes the service access point, too. The definitions for units of work SearchAction and DBQuery are straightforward. InstallDate and Status are omitted. The Id does not represented a GUID, which was a deliberate choice (compare CIM_MetricDefinition instances). Context was chosen to be the application name (CatalogSearchSystem). If the system participated in a particular business scenario, one might choose a scenario name. Only the search action writes traces. Therefore, the corresponding trace level type is attached to the search action definition. The type ID is set to the instance ID of the

CIM_TraceLevelType instance. The trace level type expresses a simple trace level encoding by means of numbers allowing for setting the trace implementation to off, errors-only and verbose (errors and informational messages).

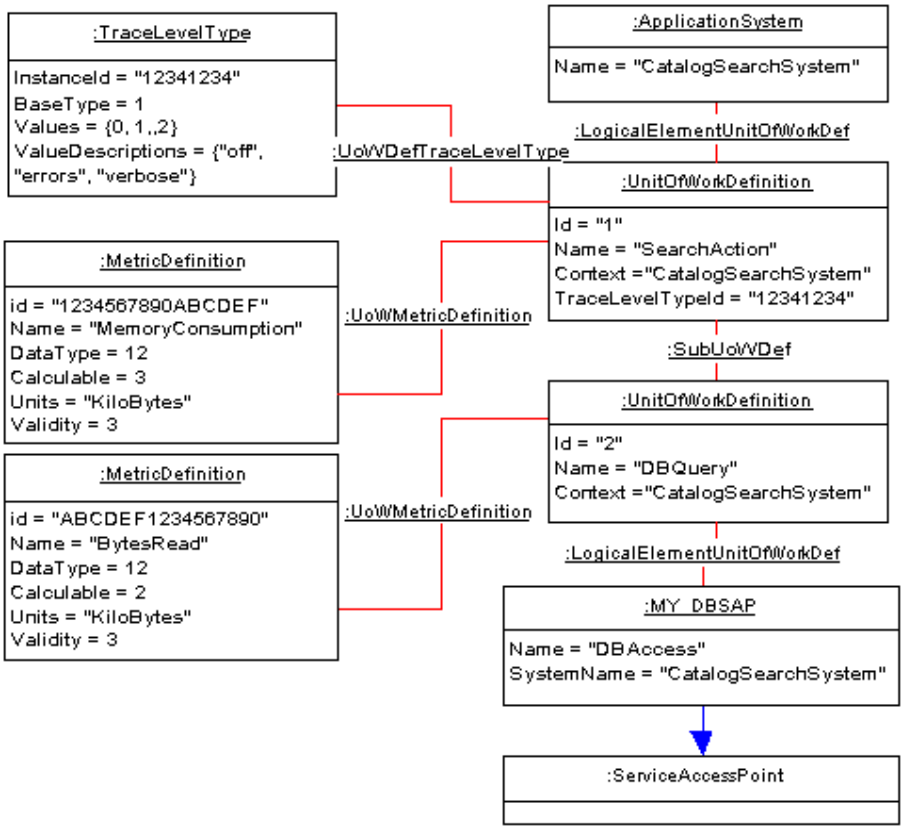

Figure 4: Definitions Instance Diagram

Since not only the elapsed time is expected to be measured, but also memory consumption and the number of bytes read for the respective units of work, these two additional metrics must be defined by instances of CIM_MetricDefinition. These two instances are attached to the respective CIM_UnitOfWorkDefinition by means of CIM_UoWMetricDefinition. The metric definitions' Caption and Description are omitted. ID is a GUID, Datatype corresponds to uint32, Validity is "atStop". MemoryConsumption.Calculable is "Non-Summable" since the amount of consumed memory does not make sense to be summed up. BytesRead.Calculable was chosen to be "Summable," since it may be valid to calculate the overall throughput to the database by summing up the values of the metric value.

The definitions are assumed to be static, such that the property values need no modification during the lifetime of the definition instances. In order to keep the example simple, no breakdown dimension is used.

4.1.2 Units of Work and Metrics

Now that the definitions are instantiated, let us look at the measurements themselves. The figure depicts only one search action with its corresponding DB query, both described as instances of UnitOfWork. The search was executed by the user

"MyUser". The action started at March 8 2002:5:45:12.2 pm (GMT) and lasted 0.5 seconds. The mutual context is "11111122222". It is an instance ID of the action and is valid for both the search and DB access. The search action was completed successfully (status = 4). If it was not yet completed, the status would be "Active" (status = 1). The memory consumption of the search was 1453 kB. Traces on the level of errors (TraceLevel = 1) are available. The nested DB query was executed at March 8 2002:5:45:12.3 pm (GMT) and lasted 0.3 seconds. Thus, the validation and presentation lasted approximately 0.2 seconds (approximately, since the impact of the measurements is not considered). The query was also completed successfully (status = 4). The number of bytes read during the query is 201 kB. The metric values for memory consumption and bytes read are stored in the respective value property of CIM_UoWMetric instances. No traces are available (TraceLevel = 1); this is according to the definition of the unit of work. The instances of CIM_LogicalElementPerformsUoW are used to attach search-units-of-work to CatalogSearchSystem and the query-units-of-work to DBAccess.

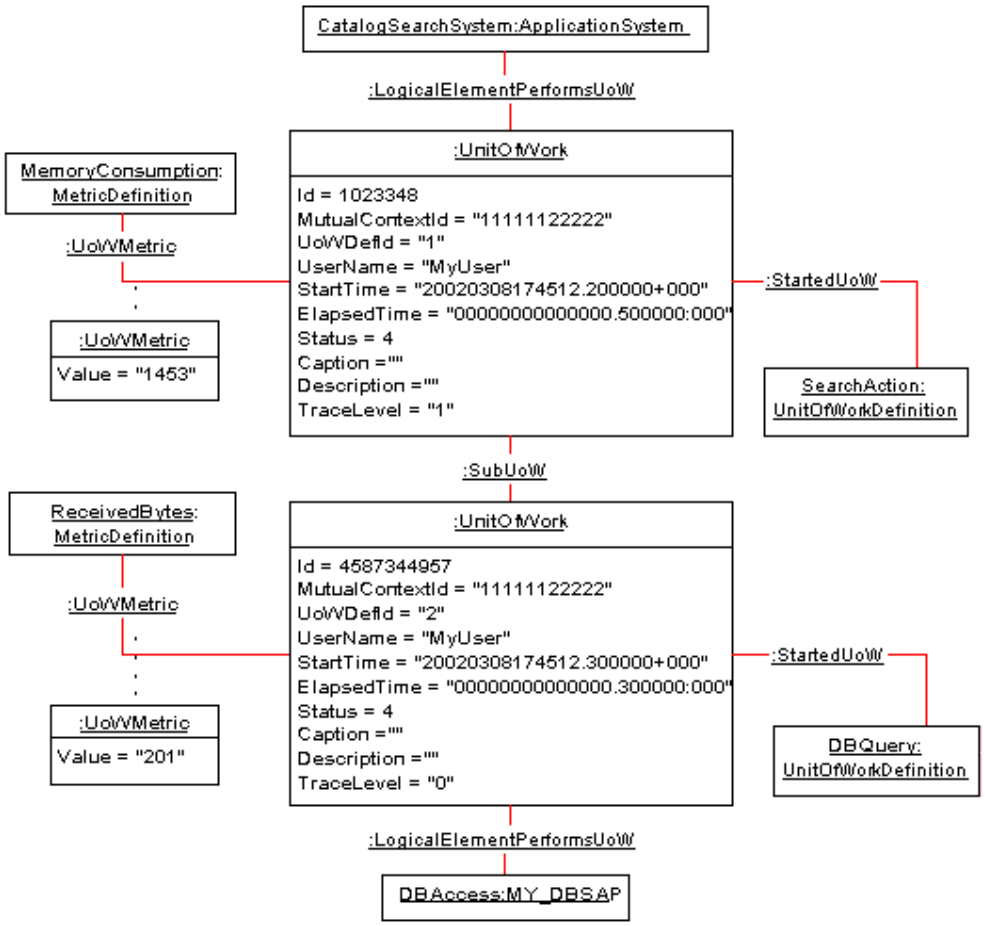

Figure 5: Metric and UnitOfWork Instance Diagram

CIM_UnitOfWork.Caption and Description have been chosen to be empty for both CIM_UnitOfWork instances.

4.2 Looking Forward, an Extended Use Case Example

The following example creates a scenario that uses the current model but includes functionality beyond the current UoW Model. It is included in this white paper because it demonstrates a real-world utilization, demonstrates the use of the current CIM Schema in cases where it must be significantly extended to solve the problem, and also provides some insight into the future directions of the Metrics working group. Note, however, that discussion of functionalities like correlation in this model is not commitments by the work group to include this functionality in the model.

The following example does not use the standard in all its facets. Its scope is a CIM description of unit of work data that adheres to the already existing, underlying data structures without imposing an entire CIM hierarchy on the application, and surfaces the data structures with minimal impact on the existing instrumentation. In addition, the management application is chosen to be custom-tailored to the model presented in this example. Look at the example as a transition model – between having no CIM Schema at all, and having a holistic model as described in the previous section.

4.2.1 Situation

A user requests a business action (e.g. search for a catalogue item). The action requires co-operation of several application systems (system A, B, C and D, running on different hosts) for successful completion. Thus, the requested business action is in fact distributed. In addition, each system may need to execute more than one local action (system A and B). These local actions are subsequent actions i.e. not nested and may also require calls to other systems (e.g. between systems A and B or B and C).

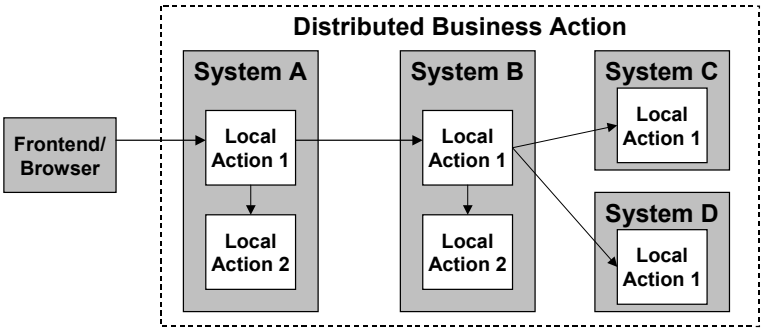

Figure 6: Scenario of a Distributed Business Action

Performance analysis and performance monitoring of the distributed business action is the ultimate goal of the application systems administrator or the application software vendor's support organization to successfully detect performance bottlenecks. Therefore, it is necessary to measure each local action and store the results of the measurement. Further, a mechanism needs to be provided that allows for the correlation of the local actions in order to re-assemble the distributed business action for monitoring or analysis purposes. The data that is generated by local measurements and for correlation purposes is to be described in a CIM Schema that allows management tools to efficiently access and work on such performance data. Although correlation is not addressed by the current Metrics Model, it seemed worthwhile to provide deeper insight into this topic.

4.2.2 Solution

The solution to the above situation is attained by using the CIM_UnitOfWork concept with some extensions for correlation. For reasons of clarity, the solution is described in two steps. The first step addresses the local measurements, the second step addresses the correlation. The solution has been implemented for SAP systems like mySAP components like the ITS (Internet Transaction Server) and is called Distributed Statistic Records (DSR). Some modelers are uneasy about CIM and historical data (which is implied by the denomination “Record”). The underlying data store is well capable of handling object lifecycle issues and the time frame data is kept is not days or weeks.

4.2.3 Local Measurements

A local action is defined as all code that is locally executed in one particular OS process. The underlying software architecture consists of several processes operating in parallel. The processes have a dedicated "type" or purpose. Due to scalability, several processes may serve the same purpose. As soon as a request is received by the application system, the request is either assigned to a free process (of the appropriate type) or, if no process is available, input to a queue for subsequent processing.

Each local action creates a so-called main record, which is an instance of a unit of work. The main record is derived from CIM_UnitOfWork and adds other statistical information about the local action. The main record consists of the properties inherited from UnitOfWork, with the addition of local context information, such as Action: name of the local action; ActionType: type of the local action and local measurements as CPUTime; QueueTime; MaxMem: the maximum amount of memory used during the action, etc. Note that the local context could have been placed in a subclass of CIM_LogicalElementPerformsUoW that is related to some CIM_Service, but modeling the entire environment, including Services, was out of scope of this modeling effort. The designers of underlying data structures also foresaw the property “Additional” for potential custom-defined parameters in the format ‘name1=value1 name2=value2”. Although it is better to clearly define the semantics of any data used in a model, it was chosen to surface this “data container” as is. A particular problem is the Id. It is no problem to locally assign a unique number. But as soon as the records/instances are to be retained in a central repository, the uniqueness is far from guaranteed. Eventually, a new key may have to be assigned on such operations, but reservation of some bits of the Id for a system identification is also a potential solution.

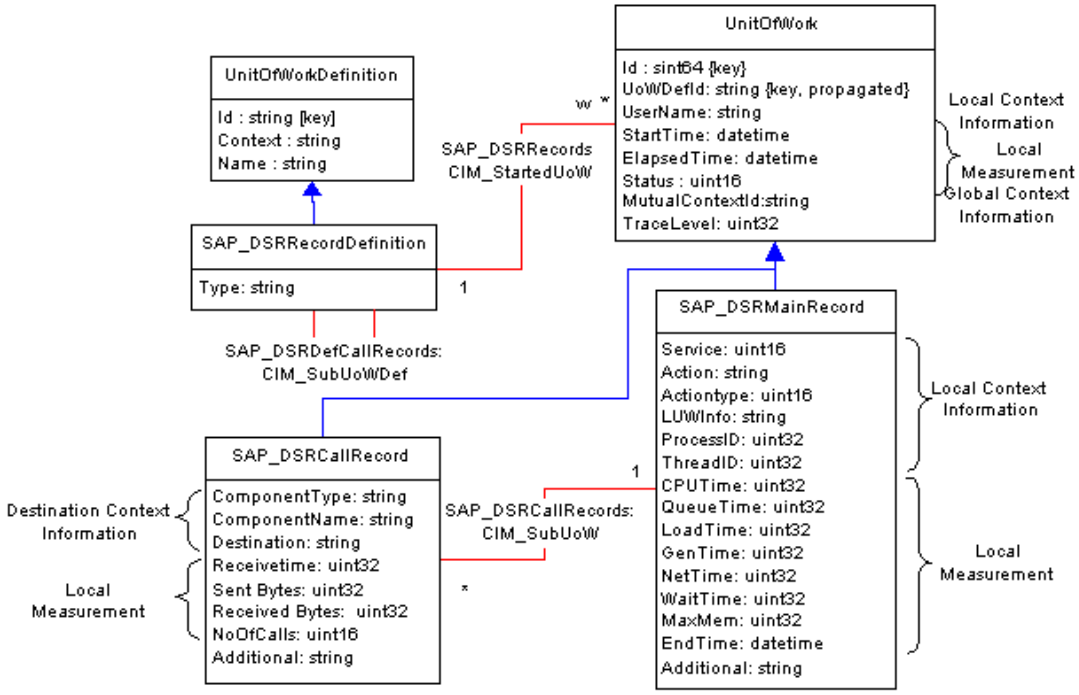

Figure 7: Using CIM_UnitOfWork and CIM_UnitOfWorkDefinition

During the course of the local action, calls to external systems may be issued. Apart from correlation that is addressed later, the information about the called system and measurements of the calls are important for performance analysis. Such information is identified in a call record. A main record can have multiple call records (or none, if no calls have been executed). The call record is the sum of all calls issued to a particular system (described by the destination context: ComponentType: type of system that is called; ComponentName: ID of the system; Destination: description of the service access point used for the calls). Note, that the destination context could have been placed in some CIM_ServiceAccessPoint associated via CIM_LogicalElementPerformsUnitOfWork, but extending the model to the environment of the units of work was not the intent of the model.

Strictly speaking, the call record could also be modeled as CIM_StatisticalInformation. But the record structure needs to have a version (which is accomplished by using CIM_UnitOfWorkDefinition), the number of calls per record is small or even equal to one (which is not the basic idea of statistical information), and finally the call record is always within the scope of a particular main record. All these arguments led to the decision to model the call record as SubUnitOfWork.

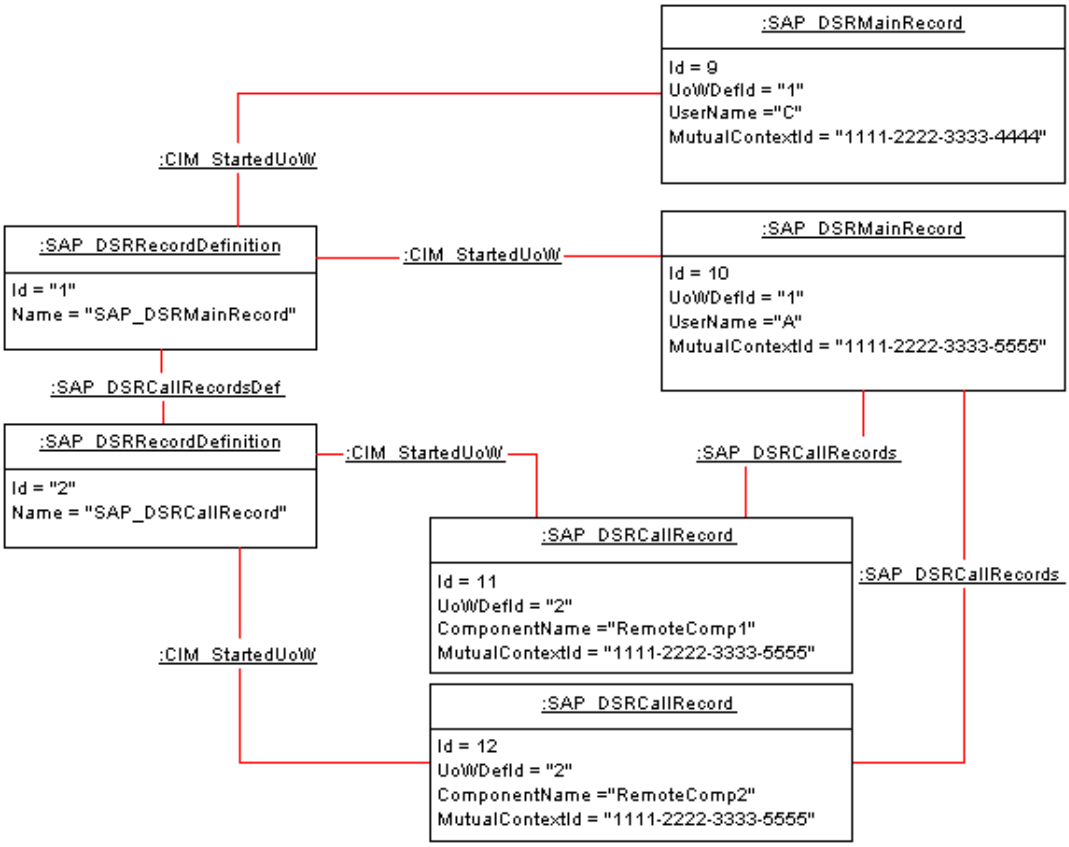

Figure 8: Instantiating CIM_UnitOfWork

In addition, the implementation supports trace levels. Since the concept of trace levels is straight forward, trace levels are not depicted to reduce the size of the example.

In order to allow the management application to benefit from CIM_MetricDefinition.Validity, CIM_BaseMetricDefinition.Calculable and to enhance standard compliance, CIM_MetricDefinition is used to add the description of the metrics that are defined in the classes SAP_DSRxxxRecord. For this purpose, SAP_DSRMetricDefinition was derived from CIM_MetricDefinition. Its instances carry the metric definition, which is associated to the appropriate UoW definition (SAP_DSRRecordDefinition). Since the metrics are already present in subclasses of CIM_UnitOfWork, information like Datatype and Name (inherited from BaseMetricDefinition) is actually duplicated. Units could be expressed by means of the appropriate qualifier or by CIM_MetricDefinition.Units.

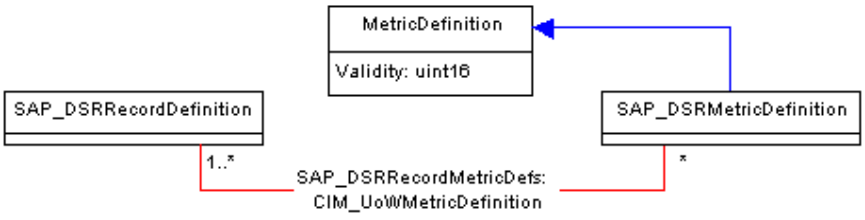

Figure 9: Using CIM_MetricDefinition

The association CIM_UoWMetric has not been used since the metric values are already published in DSRxxxRecord instances. This modeling decision was made since intended usage of the unit of work data was either the retrieval of the entire record (with all its metrics), or no record at-all. Therefore, the additional WQL query needed to also retrieve the metric values and the more complex provider implementation seemed to require too much overhead. In consequence, the management application dealing with these units of work needs to know about the deviation from the standard. Other decisions could be made in this circumstance.

The instance model shows how the definition-sub-model has been applied to the example. SAP_DSRMetricDefinition.Name corresponds to the property names defined in the DSRxxxRecords that represent the values of the metrics. For each metric property of the DSRxxxRecords a corresponding DSRMetricDefinition instance exists.

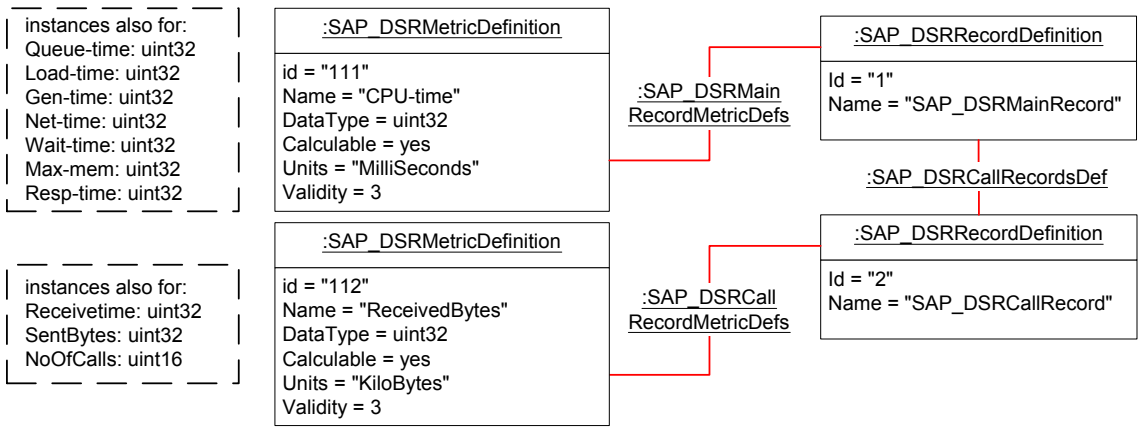

Figure 10: Instantiating CIM_MetricDefinition

4.2.4 Correlation

Now that we understand the model that describes the local measurements, the model must support the correlation of these local measurements. In order to achieve correlation, some distributed context must be defined. First, such distributed context needs to be known to each participating action, and second, it needs to be stored for subsequent aggregation. The context, the so-called ClientInfo, has been defined to hold information about the system that initiates the action (InitiatorSystem, InitiatorService: the component or function that has been addressed by the action), the user (InitiatorUserId), the first action executed (InitiatorAction, InitiatorActionType). These properties are helpful for aggregation of local actions in order to calculate statistics according to the common context of several distributed business actions. The InitiatorTransId (Trans = Transaction) is assigned to each business action once by the initiating system and allows for correlation or statistical calculations local actions of the entire business action. It also becomes visible in the MutualContextId properties of the unit of work instances.

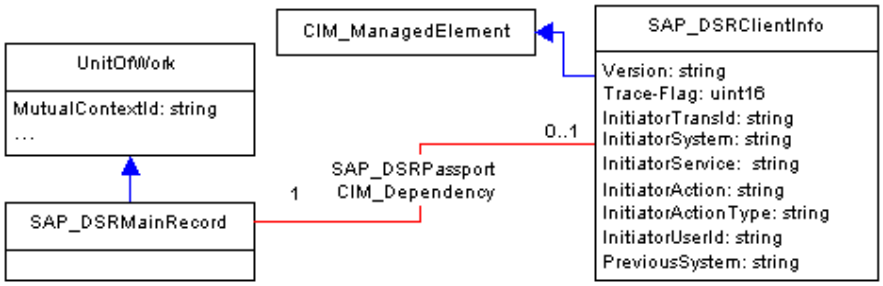

Figure 11: Inter-System Correlation

The ClientInfo is created by the first system that receives a request without ClientInfo. The InitiatorTransId is generated and all other initiator context attributes are set. Each call to another system requires that the payload of the call is enhanced with the ClientInfo. In addition, the calling system enters its system ID to the PreviousSystem-property of the ClientInfo and sets TraceFlag to the appropriate level before transmitting the ClientInfo. At the end of its local action the ClientInfo (with PreviousSystem =””) is stored together with the corresponding MainRecord. The called system receives the ClientInfo, checks whether the TraceFlag requires to write traces (remember that we resigned to depict the trace levels in the unit of work instances and definitions), and prepares the measurements of its own (local) MainRecord. On its calls to other systems, it transmits the same ClientInfo it has received, only the PreviousSystem-property has been changed to its system ID. The unchanged value of the TraceFlag allows for selective tracing of an entire business action. It corresponds to the field of ARM's correlator.

If a local action initiates subsequent local actions on the same system (each represented by a new MainRecord – compare the example in section 'Situation': SystemA.LocalAction1 initiates SystemA.LocalAction2), only the initiating MainRecord is associated with the ClientInfo (Cardinality 0..1). This is possible since the generated or received ClientInfo is internally transferred via shared memory and these subsequent local actions still store the TransId (and becomes its MutualContextId), which is enough to either relate to the ClientInfo associated with the first local action or to relate all local actions that logically share the same ClientInfo. Keep in mind that subsequent local actions also transfer the ClientInfo, in case calls to other systems are to be executed. The local actions could be associated via CIM_SubUoW (not depicted in the figures). In contrast, the ClientInfo resolves the necessary cross-namespace CIM_SubUoW associations that would be needed between related MainRecords on different systems. This modeling decision allows one to access all needed information local to one provider, and represents a solution that is less abstracted from implementation issues. The properties of the ClientInfo class may be considered as the externalized properties of a cross-namespace CIM_SubUoW association subclass.

4.2.5 Lessons Learned in this Model

1. The above example mixes semantics expressed on the level of the model (e.g. metric properties already defined in the derived UoW-classes), and semantics expressed on the instance level (UoW and metric definitions). It is a compromise between the simplest programmatic access by the management

applications that know about these specific UoW subclasses, as well as simpler provider implementation, and the standard.

2. In order to provide powerful correlation information, the additional definition of such information (s. SAP_DSRClientInfo) seems to be necessary. However, discussion is still in progress.

3. Some of the context properties defined in SAP_DSRxxxRecords implicitly refer to systems, processes and services. Usage of CIM_LogicalElementPerformsUnitOfWork would require explicit modeling of all otherwise implicitly referred entities, which, at the time, was beyond the scope of the example. So, in a first step, a model that reduces the number of entities that need to be modeled and implemented is a feasible solution.

4. Sometimes #1 and #3 are referred to as denormalization of a model.

Appendix A – Change History

Version 0.1 (Draft)	October 12 2002	Preliminary version that incorporates some additions to V2.6 (concept of BaseMetrics excluded, CR 819 (Mutual Context ID) and CR 818 (Trace Level) included), CR 820 (Base Metrics) is not included.
Version 0.2	March 25, 2003	Added Base Metric-specific information
Version 0.3	March 28, 2003	Some changes after initial discussion in Metrics Model Extensions subgroup of the Applications WG

Appendix B – References

CIM Core and Common Models - Versions 2.0 through 2.7 - Downloadable from http://www.dmtf.org/standards/standard_cim.php

Common Information Model (CIM) Specification, V2.2, June 14, 1999 - Downloadable from http://www.dmtf.org/standards/documents/CIM/DSP0004.pdf

DMTF Specifications - Approved Errata - Downloadable from http://www.dmtf.org/standards/standard_cim.php

Desktop Management Interface (DMI) - Downloadable from http://www.dmtf.org/standards/standard_dmi.php

Internet Engineering Task Force (IETF) - MIBs and Work Group information at http://www.ietf.org

Application Response Measurement (ARM) 3.0 Java Binding, Open Group Technical Standard - Downloadable from http://www.opengroup.org/management/arm.htm

Application Response Measurement (ARM) 2.0, Open Group Technical Standard - Downloadable from http://www.opengroup.org/management/arm.htm or from www.cmg.org