Measuring Application Response Times with the CIM Metrics Model

Alexander Keller[1], Andreas Köppel[2], and Karl Schopmeyer[3]

[1] IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598, USA
alexk@us.ibm.com
[2] SAP AG, Infrastructure Management (CCMS), Neurottstr. 16, 69190 Walldorf, Germany
andreas.koeppel@sap.com
[3] The Open Group
k.schopmeyer@opengroup.org

Abstract. We describe new extensions to the CIM Metrics Model. While the previous version of the model provides a means to carry out response time measurements against an application, the scope of the work described in this paper is to define schema extensions capable of measuring and tracing distributed transactions. The model has been developed by the Application Working Group of the Distributed Management Task Force (DMTF) in which the authors actively participate. The extensions described in this paper have been recently adopted by the CIM Technical Committee and are part of the new version 2.7 of the CIM schema. Two detailed examples illustrate the applicability of the Metrics Model to real-life measurement scenarios in distributed computing environments.

1 Introduction and Motivation

The DMTF Common Information Model (CIM) [3,4,6] is a standardized framework for the management of systems, software, users, networks and more that relies on the basic structuring and conceptualization techniques of the object-oriented paradigm. CIM provides a management information model to establish a common conceptual framework for describing the managed environment.

The CIM Metrics Model is one of the CIM Schemas. It was originally developed within the CIM DAP (Distributed Application Performance) Working Group, which was later renamed Metrics Working Group. Its goal was to develop a CIM schema for measurement information about a **Unit of Work (UoW)**, which is measured with tools according to the *Application Response Measurement (ARM)* standard. Several ARM usage examples have been published: [7] describes an example of using ARM and CIM units of work to measure and publish response times of EJB applications; [9] proposes an approach for correlating ARM measurements among distributed units of work.

In the year 2000, the Applications Working Group began to develop a model specifically for the management of applications at runtime (the initial applications model deals primarily with software distribution); we have reported on the first results of this work in [10]. One of the specific objectives of this work is the modeling of metric data, such as counters and gauges. It became apparent that the schema of the Metrics group and the work of the Applications Working Group on metrics had common objectives, which led

M. Feridun et al. (Eds.): DSOM 2002, LNCS 2506, pp. 66–81, 2002.

to the inclusion of the Metrics Model into the scope of the Applications Working Group. The principal goals of the Metrics Model are as follows:

1. Defining a model for representing UnitOfWork metrics and their definitions; an instance of a metric exists 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. Showing the currently pending or recently finished actions,
3. Publishing the semantics of the measurement data, i.e., providing a definition for the management system so that it is able to understand what the data means,
4. Relating the measurements to the entity (application system, service, component, etc.) that executed the measured operation or action,
5. Relating the measurements hierarchically as well as sequentially (correlation).

In the following sections, we describe the design of the Metrics Model and how it addresses these requirements. Section 2 introduces the underlying Unit of Work concept and examines the relationship of the CIM Metrics Model to the ARM standard. Section 3 presents the Metrics Model and the new extensions, the target of our effort. Special emphasis is put on how this model can be applied in practice: We describe in detail two typical usage scenarios for the Metrics Model in section 4. Section 5 concludes the paper by discussing the lessons learned during the design of the model, and presents current items of the working group.

2 The Unit of Work Concept

Generally speaking, a **Unit of Work (UoW)** is an individually distinguishible set of executable procedures defined in a (potentially distributed) execution environment. Originally defined for transaction response time measurement, the Unit of Work concept has been 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), network round trip delays, or even workflows. The term "unit of work" is used instead of "transaction" because the latter term may imply a specific behavior and, as the list above shows, there are many possible units of work beyond transaction processing. Any unit of work measurement has two primary properties:

1. The time it took to complete the unit of work, or elapsed time if it is still executing.
2. The status of the unit of work: Active, Suspended, Completed Good, Completed Failed, Completed Aborted, Completed (with unknown state).

In addition, it is useful to understand if a unit of work depends on another one, such as one unit of work 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. 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.

2.1 Relationship of the CIM Metrics Model to the ARM Specification

The Application Response Measurement (ARM) standard [1,2] has been developed under the auspices of the Open Group as a way to instrument applications for measurement data about transaction response times from a client perspective. The CIM Metrics model's objective is the definition of the information objects that represent units of work – a generalization of the initial problem of transaction response time measurement. ARM and the CIM Metrics model are based on the same fundamental concept; thus, it is not surprising that the Metrics Model can be used to represent data measured with ARM. While ARM and the CIM Metrics Model were developed as components of a common solution to the problem of capturing UoW information, users are free to use any other appropriate instrumentation to populate the CIM Metrics Model. The following two sections briefly provide some background on ARM and compare ARM and the CIM Metrics Model.

The Application Response Measurement (ARM) API. Introduced in 1996, ARM 1.0 and, later, 2.0 [1] specify C APIs while ARM 3.0 [2] describes Java interfaces for an ARM agent. These standardized APIs can be used to capture performance information for transactions or any programming function where the time-to-completion is important. ARM does not define the characteristics of the resulting data, or the means for communicating this data to a management system. Conceptually, the ARM interfaces are very simple (*Start_Transaction*, and *Stop_Transaction*, with a few other supporting interfaces). In its new version 3.0, ARM interfaces address the following two situations:

1. The ARM agent is triggered by the instrumented application to carry out the measurements (interface *ArmTransaction*),
2. The ARM agent receives measurements completed by the agent (*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 "parent correlator") of the current action is stored with the measurement data. ARM 3.0 also allows enhanced transaction measurements (and its inherent response time metric) with up to seven additional metrics. Therefore, the interfaces *ArmTranReportWithMetrics* and *ArmTransactionWithMetrics* are defined to extend the transaction interfaces. ARM metrics fall into four pre-defined categories: gauges, counters, non-calculable numeric values, and strings. 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*).

Comparing the CIM Metrics Model and ARM. ARM defines an API that consists of several interface methods that are supposed to be used in a specific sequence. On the other hand, *CIM_UnitOfWork* and its associated classes define a data model by abstracting from the metrics and the actual instrumentation. Although the ARM specification also provides diagrams that describe the data model implemented by the API, the model itself has no formal data representation. Table 1 lists the CIM classes and their ARM

Table 1. Classes of the CIM Metrics Model and ARM

CIM Metrics 2.7	ARM 3.0
CIM_UnitOfWork	ARMTransaction ARMTransactionWithMetrics ARMTranReport ARMTranReportWithMetrics
CIM_UnitOfWorkDefinition	ARMTranDefinition
CIM_MetricDefinition	ARMMetricDefinition
CIM User Schema	ARMUserDefinition
CIM_UoWMetric	ARMMetric ARMTranReportWithMetrics.MetricValues
CIM_LogicalElementPerformsUoW	ARMSystem ARMSystemId
CIM_SubUoW	ARMCorrelator
CIM_LogicalElementUnitOfWorkDef	N/A

equivalents. There is no semantic gap between the ARM API and CIM, although the ARM classes target transactions while the scope of the CIM Metrics Schema is broader. Although the concepts behind the class *CIM_UnitOfWork* match ARM very well, some conceptual differences can be identified. The fact that the CIM Metrics Model describes the elements of a measurement itself contrasts with the API defined by ARM as follows:

While the purpose of ARM is to define the data access through a few well-defined methods, the Metrics Model itself does not provide any data access interface, because, in CIM, management data is supposed to be surfaced by a CIM Object Manager (CIMOM) whose access operations are generic (cf. [5]). The most important difference concerns the way how metrics and their types are defined: While ARM is restricted to 10 predefined metric types, CIM allows the user to define arbitrary metric types. In addition, an ARM transaction may carry a maximum of seven metrics, while CIM does not restrict the number of metrics associated with a unit of work. To sum up, CIM introduces a concept of metrics that is more flexible and adaptable to the requirements of a specific environment than ARM.

3 The CIM Metrics Model

3.1 The Core Elements: Unit of Work, UoW Definitions, and Sub-units of Work

The central class of the CIM Metric Model, depicted in figure 1, is *CIM_UnitOfWork*. It represents an individual action or operation. It has an identity (which is its key) and provides contextual information (e.g., the UserName) and, most importantly, the response time measurement values together 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. Thus, *CIM_UnitOfWork* instances are associated via *CIM_StartedUoW* to their corresponding definition (*CIM_UnitOfWorkDefinition*). The unit of work instance can also be related to a logical element (i.e. some instance of *CIM_System*, *CIM_Service*, etc.) that executes/has executed the unit of work. This is done via the association *CIM_LogicalElementPerformsUoW*. An important feature of the model is the association *CIM_SubUoW*: It allows the correlation between various instances of *CIM_UnitOfWork* that are hierarchically related. A high-level unit of work

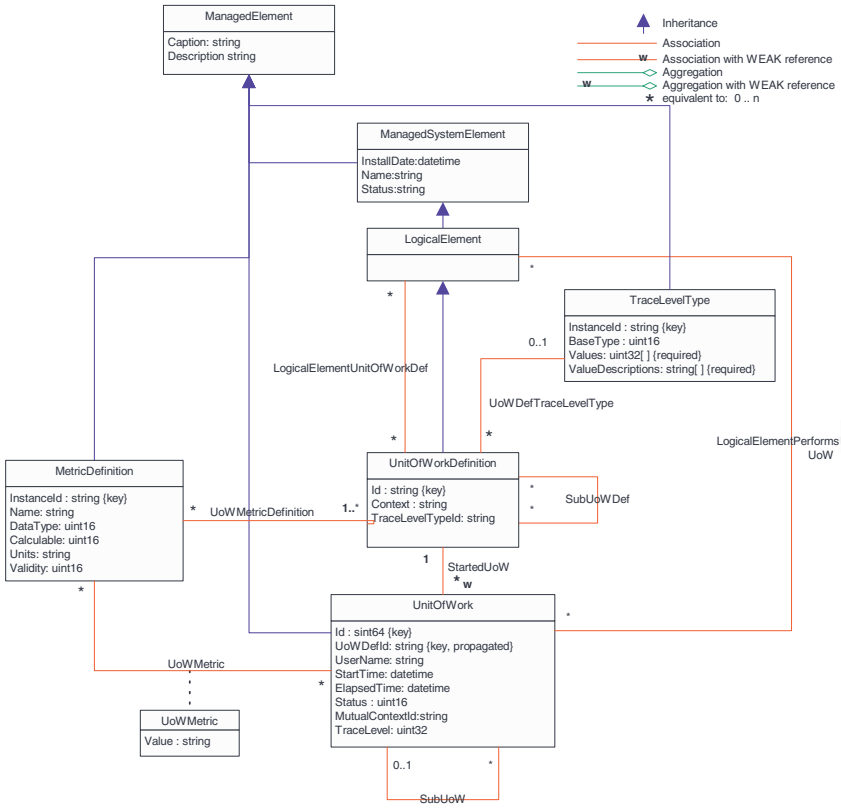

Fig. 1. The Unit of Work Part of the CIM Metrics Model 2.7

can be broken down into 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. There are ongoing discussions within the ARM community to define additional semantics indicating if a sub-transaction is blocking or non-blocking.

CIM_SubUoW is not intended to model sequential relationships of units of work, but represents blocking and non-blocking hierarchical calls. If the unit of work hierarchies are defined and known before the execution of the action or operation, *CIM_SubUoWDef* expresses the anticipated relationships between the instances. A management application could use such information to determine incorrect execution paths or to create a graphical representation of the measurements. Note that today, an instance of a sub-unit of work can be associated to only one parent. The reason is that 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.

3.2 Correlation with *CIM_UnitOfWork.*`MutualContextId`

The correlation context of *CIM_UnitOfWork* instances is provided by *CIM_SubUoW* association instances. Thus, the mutual context of two *CIM_UnitOfWork* instances has to be retrieved by following the associations between these instances, i.e., the mutual context is not named explicitly. It may be needed for two situations:

- Displaying all *CIM_UnitOfWork* instances that participate in an action as a whole.
- Direct retrieval of *CIM_UnitOfWork* instances that participate in the same context by using one query expression.

To facilitate these operations, we have introduced the property `MutualContextId`. It is not supposed to replace *CIM_SubUoW* association instances, but should support better identification and correlation of distributed actions.

Cross-namespace associations are an important issue when using *CIM_SubUoWDef* and *CIM_SubUoW* for correlation. They arise if the associated instances reside within different namespaces (e.g., on different systems), i.e., the instances of *CIM_UnitOfWork* are provided separately and thus need to be related by a cross-namespace association. Current CIMOM implementations cannot resolve an association having a reference that points to another namespace (neither local nor remote). Possible workarounds are:

- Organizing the CIM providers in a way that they operate against one namespace.
- Using a separate correlation information class that carries the necessary correlation properties (see the example described in section 4.2 for details). A management system could then access this correlation information and provide the associations.

3.3 UoW Metrics and Metric Definitions

CIM_UnitOfWork can be associated with metrics, whose semantics and data type can be defined in the class *CIM_MetricDefinition*. The metric definition is always defined in conjunction with at least one definition of a unit of work (*CIM_UoWMetricDefinition*). In analogy to the unit of work definition, the metric definition provides additional information to a management application to understand the semantics and usage of the metric (Metadata). The value of a metric is held in the `Value` property of the *CIM_UoWMetric* association. Providing the metric value by means of this association avoids cluttering the model. An alternative would have been to model the value of the metric as an additional CIM metric class. However, this additional class would still need the association to *CIM_MetricDefinition* and an additional association to *CIM_UnitOfWork*. This would result in many instances to obtain just a single value.

The reasons why the definition is kept separate from the values of unit of work and metrics are as follows:

1. This model provides flexibility because its class definitions need not be changed to provide new types of units of work. One only needs to create new instances. Since there may be many different metric types, this clearly is the 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 shared among many different *CIM_UnitOfWorkDefinition* instances.

An alternative approach consists in defining qualifiers of the unit of work class. This approach has the same shortcomings of item 1 above, and a few additional ones. For example, more than one qualifier would be needed to describe the definition. In addition, there is no way to show the relationships between these qualifiers. Yet another alternative is to include the properties of the unit of work into its definition. This, however, results in mixing data with metadata and duplication of definitional information in every instance.

3.4 Facilitating Traces

CIM_UnitOfWork.`Status` and *CIM_UnitOfWork*.`ElapsedTime` may be indicators of faulty system behavior or performance bottlenecks. However, they are usually insufficient to determine the cause of a fault or a performance bottleneck. More detailed information is often available from traces written during the execution of a unit of work. An administrator needs to know whether traces for some particular unit of work instance are available. The trace level is the indicator whether traces are available for a given unit of work instance. It also defines the granularity of the traces produced. However, it does not indicate where the traces are found and how they are accessed. Thus, in addition to the properties `TraceLevelType` and `TraceLevel` of the classes *CIM_UnitOfWorkDefinition* and *CIM_UnitOfWork*, a new class *CIM_TraceLevelType* and an association *CIM_UoWDefTraceLevelType* are included in the model. Since the semantics of the trace level are usually implementation dependent, a management application may also need to distinguish between different semantics (the trace level type) as well as the actual encoding of the applied types (i.e., whether the level is represented by a bitmap or an enumeration and what the different possible levels mean). Note that the association *CIM_UoWDefTraceLevelType* is not attached to *CIM_UnitOfWork* since it seems prohibitive to burden the potentially numerous *CIM_UnitOfWork* objects with additional associations. A desirable side effect of this approach is that it prevents the implementation of different trace level encodings for *CIM_UnitOfWork* instances having the same definition, and thus ensures consistency. Section 4.2 demonstrates the applicability of this approach for defining traces in a distributed context.

4 Examples of Using the Metrics Model

4.1 Using the CIM Metrics Model to Measure Nested Units of Work

To demonstrate the usage of the Metrics Model, a simple example has been chosen that uses all the features of the model. As depicted in figure 2, we assume a simple application system (Catalog Application System) that provides search functions for a catalog stored in a database. When a catalog search request is issued against the system, the request is validated (format of the query) and handed over to a database service access point (DB SAP) that executes the DB query. Finally, the results of the query are prepared for presentation and returned to the requestor. The system is able to work on several requests in parallel.

Administrators need information on blocking requests as well as detailed per-request performance information to optimally tune their catalog system and database. They want to measure the entire search operation, including its duration (elapsed time) and memory consumption. Since 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.

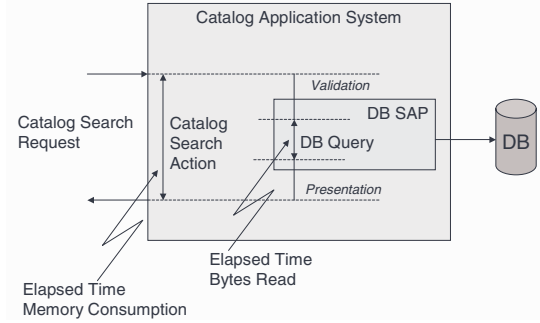

Fig. 2. Scenario: Nested Units of Work

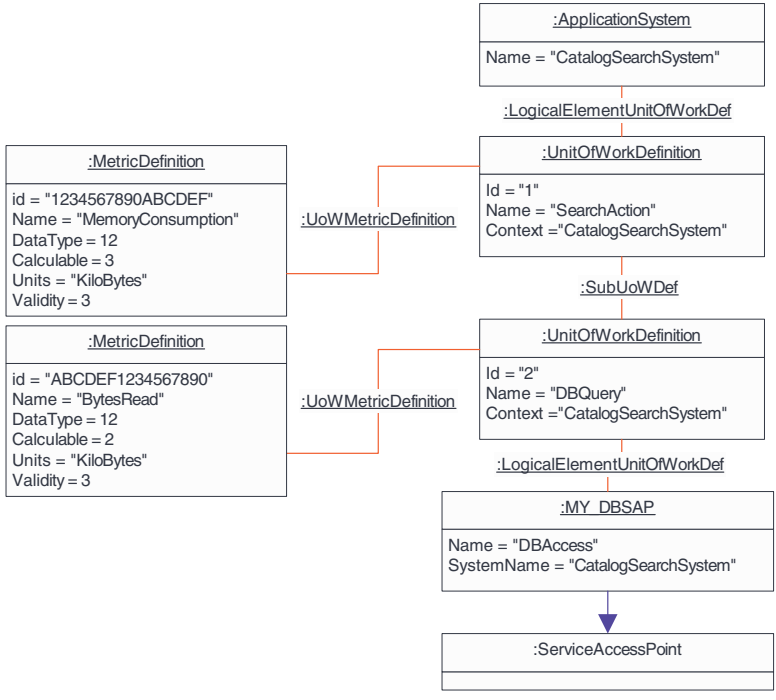

Fig. 3. Instance Diagram of the Definition Objects

Setting up the Measurements: Unit of Work Definitions and Metric Definitions. For simplicity, the classes of the Metrics Model have been used "as is". It is up to

the designer to decide whether deriving more specific subclasses are needed. However, subclassing from *CIM_ServiceAccessPoint* is necessary since this class is abstract and, thus, 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 (DBAccess). The "Hosted by" relationship is not depicted in figure 3, but DBAccess is assumed to be hosted by CatalogSearchSystem. The application may be described by different instances of *CIM_LogicalElement* or its subclasses *CIM_SoftwareFeature* or *CIM_SoftwareElement*.

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 unit of work 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. The definitions for the units of work SearchAction and DBQuery are straightforward: In this example, the `Id` does not represent a GUID; `Context` is set to the application name (CatalogSearchSystem). We have omitted the properties `InstallDate` and `Status`.

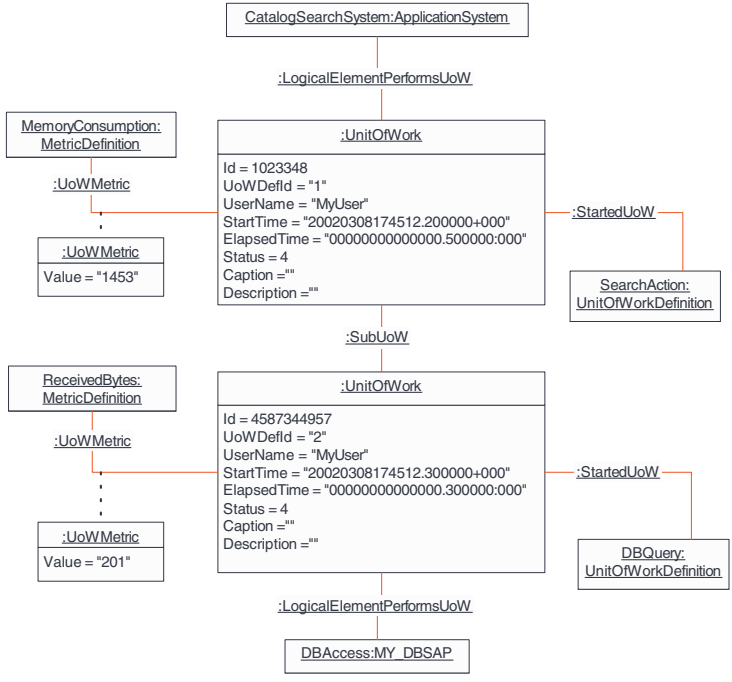

Fig. 4. Instance Diagram of the Metric and UnitOfWork Objects

Since we do not only want to measure the elapsed time, but also memory consumption and the number of bytes read for the respective units of work, these two additional metrics must be defined as instances of *CIM_MetricDefinition*. These two instances are attached to the corresponding *CIM_UnitOfWorkDefinition* by means of *CIM_UoWMetricDefinition*. The metric definitions' `Caption` and `Description` are omitted for the sake of brevity. `ID` is a GUID, `Datatype` corresponds to uint32 (encoded as an enumerated value "2"), `Validity` is "atStop" (encoded as "3"). The `Calculable` property of the MemoryConsumption object is "Non-Summable" (encoded as "3") since it does not make sense to aggregate the amount of consumed memory. BytesRead.`Calculable`, on the other hand, was chosen to be "Summable" (encoded as "2") because it may be valid to calculate the overall throughput of the database by summing up the values of this metric. The definitions are assumed to be static, i.e., the property values are not modified during the lifetime of the definition instances.

Obtaining the Results: Units of Work and Metrics. Figure 4 depicts the measurement data for one search operation with its corresponding DB query. Both are described as instances of *CIM_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 search action was completed successfully (`Status` = 4). If it was not yet completed, its status would be "Active" (`Status` = 1). The memory consumption of the search was 1453 kB. The nested DB query was executed on March 8 2002 at 5:45:12.3 PM (GMT) and lasted 0.3 seconds. Thus, the validation and presentation lasted approximately (since the impact of the measurements themselves is not considered) 0.2 seconds. 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 each stored in the `Value` property of the *CIM_UoWMetric* instances. The instances of *CIM_LogicalElementPerformsUoW* are used to attach search-units of work to CatalogSearchSystem and the query-units ofwork to DBAccess. *CIM_UnitOfWork*.`Caption` and `Description` have been chosen to be empty for both *CIM_UnitOfWork* instances.

4.2 An Advanced Use Case: Distributed Statistic Records

The following example describes a more advanced scenario that fully exploits the functionality offered by the Metrics Model for solving the problem of correlating statistical measurements obtained from a real-world distributed application. This example demonstrates how a developer may define extensions to address specific requirements. It uses the existing CIM Metrics Model without requiring the use of other CIM schemas so that its impact on existing instrumentation is minimal.

The Scenario. A user requests a business action (e.g., a search for a catalogue item) that requires cooperation of several application systems (systems A, B, C and D, each running on different hosts). In addition, each system may need to execute more than one local action (system A and B). These local actions are sequential actions, i.e., not nested and may also require calls to other systems (e.g., between systems A and B or B and C).

Performance monitoring and analysis of the distributed business action is the ultimate goal of the application system's administrator or the application software vendor's support organization to 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 correlates local actions to re-assemble the distributed business action for monitoring or analysis purposes. The data, generated by local measurements and subsequently correlated, is to be described in a CIM model that allows management tools to efficiently access and process such performance data.

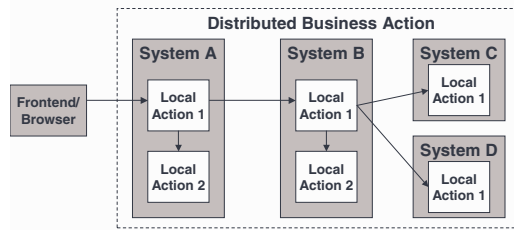

Fig. 5. Scenario of a Distributed Business Action

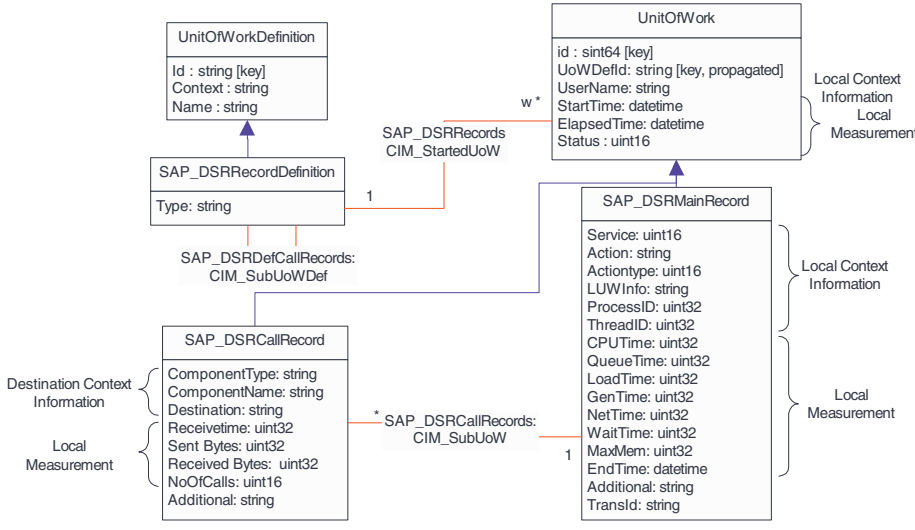

Fig. 6. Using *CIM_UnitOfWork* and *CIM_UnitOfWorkDefinition* for DSRs

The solution to the above situation is to use the Unit of Work concept with some extensions for correlation, as described in section 3.2. For clarity, the solution is described in two steps: Step one addresses the local measurements, step two addresses correlation. This solution has been implemented for SAP systems, i.e., mySAP components such as Internet Transaction Server (ITS) and is called **Distributed Statistic Records (DSR)**. The underlying data store is capable of handling object lifecycle issues and the time frame during which data is kept is typically between an hour and a day, although it can be configured to keep data for longer time periods.

Local Measurements. A local action is defined as all code locally executed within one OS process. The underlying software architecture consists of several processes

operating in parallel. The processes have a dedicated "type" or purpose and may be executed concurrently. When a request is received by the application system, the request is either assigned to a free process (of the appropriate type) or queued for subsequent processing.

Each local action creates a **Main Record**, which is an instance of a unit of work (see figure 6). The main record is derived from *CIM_UnitOfWork* and adds further statistical information for the local action. It extends the properties inherited from *CIM_UnitOfWork* by local context information such as `Action` (name of the local action), `ActionType` (type of the local action) and local measurements such as `CPUTime`, `QueueTime`, `MaxMem` (maximum amount of memory used during the action), etc. The underlying data structures of the SAP system define the property `Additional` for potential custom-defined parameters in a name-value pair format. A particular problem is the `Id`: While it is not a problem to locally assign a unique number its uniqueness is not guaranteed in a scenario where the records/instances are to be retained in a central repository. Eventually, a new key may have to be assigned to such operations, but reserving some bits of the `Id` for system identification is also a potential solution. Figure 7 depicts an instance diagram of the various record types.

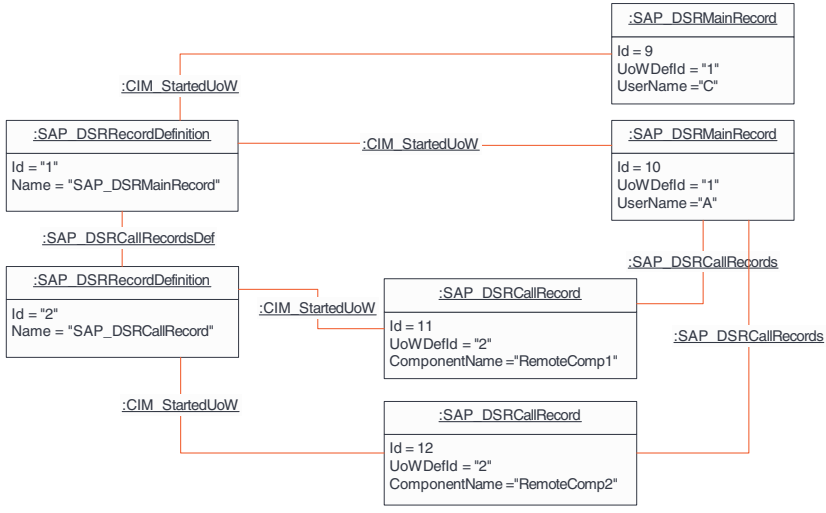

Fig. 7. Instance Diagram for Main Records and Call Records

During the course of the local action, calls to external systems may be issued. Apart from correlation (described below), the information about the called system and measurements of the calls are important for performance analysis. Such information is identified in a **Call Record** and represented by the class *SAP_DSRCallRecord*. A main record *SAP_DSRMainRecord* can have multiple call records (or none if no calls have been executed). The call record logs the calls issued to a particular system: This is

described by the following destination context information: `ComponentType` (type of system called), `ComponentName` (ID of the system), `Destination` (service access point used for the calls). Alternatively, the destination context could have been placed in some *CIM_ServiceAccessPoint* class, associated via *CIM_LogicalElementPerformsUOW*; however, 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*. However, the record structure needs to have a version (which is accomplished by using *CIM_UnitOfWorkDefinition*); in addition, 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 *CIM_UnitOfWork*.

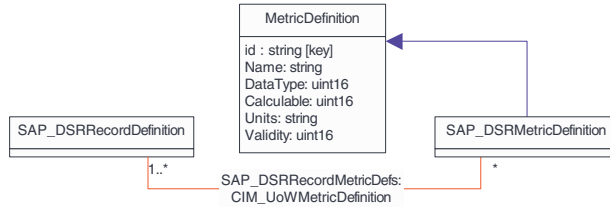

Fig. 8. Using CIM_MetricDefinition

To allow the management application to benefit from *CIM_MetricDefinition*.`Validity` and `Calculable`, *CIM_MetricDefinition* is used to add the description of the metrics defined in the classes *SAP_DSRxxxRecord* (where 'xxx' stands for either 'Main' or 'Call'). For this purpose, *SAP_DSRMetricDefinition* was derived from *CIM_MetricDefinition* (see figure 8). 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` is actually duplicated. Units may be expressed either by means of an appropriate qualifier or by *CIM_MetricDefinition*.`Units`.

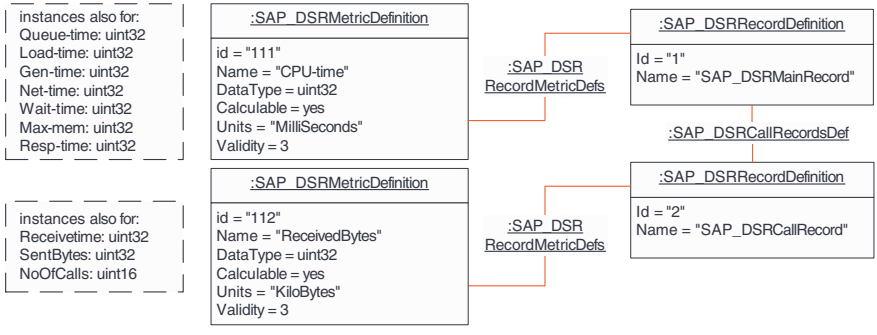

Fig. 9. Instantiating the DSR subclasses of CIM_MetricDefinition

The association *CIM_UoWMetric* has not been used since the metric values are already published in *SAP_DSRxxxRecord* instances. This modeling decision was made because the intended usage of the unit of work data is either the retrieval of the entire record (with all its metrics) or no record at all. Therefore, an additional query, specified in the *CIM Query Language (CQL)*, would also have to retrieve the metric values, which adds additional overhead. Consequently, a management application dealing with these units of work would have to be aware of this deviation from the standard. The instance diagram, depicted in figure 9, shows how the definition-sub-model has been applied to the example. *SAP_DSRMetricDefinition*.`Name` corresponds to the property names defined in the *SAP_DSRxxxRecord*s that represent the values of the metrics. For each metric property of the *SAP_DSRxxxRecord*s, a corresponding *SAP_DSRMetricDefinition* instance exists.

Correlation of distributed Measurements. Now that the model for defining local measurements has been described, we will focus on how to correlate these local measurements across several distributed systems. To achieve correlation, a distributed context needs to be established for each participating action and stored for subsequent aggregation. The context, captured in the newly defined class *SAP_DSRClientInfo*, is defined to hold information about the system that initiates the action: `InitiatorSystem`, `InitiatorService` (the component or function that has been adressed by the action), the user (`InitiatorUserId`), the first action executed and its type (`InitiatorAction`, `InitiatorActionType`). These properties are helpful for aggregating local actions to calculate statistics according to the common context of several distributed business actions. The `MutualContextId`, discussed in section 3.2, is assigned to each business action only once by the initiating system and ties the results of the statistical calculations for local actions back to the overall business action.

The new class *SAP_DSRClientInfo* is instantiated by the first system that receives a request without such client information. In addition, a newly defined association *SAP_DSRPassport* is instantiated, whose purpose is to connect the new *SAP_DSRClientInfo* instance to the *SAP_DSRMainRecord* instance. Furthermore, the `MutualContextId` property and all other initiator context attributes are set. Each call to another system requires that the payload of the call is to be extended with the client information. Finally, the calling system enters its system ID into the `PreviousSystem` property of *SAP_DSRClientInfo* and sets `TraceLevel` (introduced in section 3.4) to the appropriate level before transmitting the client information. At the end of its local action, the client information (with `PreviousSystem`="") is stored together with the corresponding *SAP_DSRMainRecord*. The called system receives the client information, checks whether the `TraceLevel` requires that traces be written and prepares the measurements of its own (local) *SAP_DSRMainRecord*. On its calls to other systems, it transmits the same *SAP_DSRClientInfo* it has received and changes the value of the `PreviousSystem` property to its own system ID. The unchanged value of the `TraceLevel` allows the consistent tracing of an entire business action. It corresponds to the ARM correlator field.

If a local action initiates subsequent local actions on the same system (each action is represented by a new *SAP_DSRMainRecord*, cf. the scenario depicted in figure 5, where SystemA.LocalAction1 inititiates SystemA.LocalAction2), only the initiating

MainRecord is associated with the client information (Cardinality 0..1). This is possible because the generated (or received) client information is transferred internally via shared memory. Subsequent local actions store the `MutualContextId`, which is sufficient to either relate to the client information associated with the first local action or to relate all local actions that logically share the same *SAP_DSRClientInfo* instance. Note that subsequent local actions transfer the client information if calls to other systems are executed. Instead of using *CIM_SubUoW* to associate local actions, *SAP_DSRClientInfo* holds this information. This eliminates the problem of resolving cross-namespace *CIM_SubUoW* associations that would be needed between related *SAP_DSRMainRecord* instances on different systems. This modeling decision allows the access of all needed information local to one CIM provider and is independent from CIMOM implementation issues (such as the traversal of cross-namespace associations, mentioned in section 3.2). Conceptually, the properties of the *SAP_DSRClientInfo* class can be thought of as the externalized properties of a cross-namespace subclass of the *CIM_SubUoW* association.

5 Lessons Learned and Outlook

In this paper, we have described the Metrics Model and its extensions, which have been adopted for CIM version 2.7. After comparing the Metrics Model with the most widely used standard for measuring application response times, the ARM API, we have described the design decisions that went into this model. To illustrate the applicability of the model to real-life environments, we have described two typical usage scenarios that have been implemented by the authors: the first scenario deals with determining application response times for nested transactions, while the second, more complex scenario describes distributed statistic records. This advanced use case leverages recent additions to the Metrics Model, such as setting up traces and establishing a mutual context. It also presents several examples how the Metrics Model may be enhanced with environment-specific extensions.

During this work, it was recognized that the concepts behind the Metrics Model are not confined to measuring response times of distributed applications, but apply to a wide range of managed elements (devices, systems, networks, SLAs, etc.). Thus, it is the intention of the Applications Working Group to extend the model described in this paper so that it provides a basis for the generic modeling of metrics in a managed environment. This includes the definition of a mechanism for dynamically (i.e., at runtime) associating both metrics and their definitions with any managed element. Similar to the concept of UoW Definitions and Units of Work, described in this paper, such a mechanism allows the dynamic creation and enumeration of all the metric instances for a given metric definition. The emerging Service Level Agreement model, which is currently being developed by the CIM Policy Working Group, is another area where the concepts behind the Metrics Model may be applied.

Acknowledgments. The authors would like to thank the members of the DMTF Application Working Group for helpful discussions and continuous advice, and their commitment to making this work succeed.

References

1. Systems Management: Application Response Measurement (ARM) API. Open Group Technical Standard, Document Number: C807, The Open Group, July 1998.
2. Application Response Measurement, Issue 3.0 – Java Binding. Open Group Technical Standard, Document Number: C014, The Open Group, October 2001.
3. 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.
4. Common Information Model (CIM) Version 2.2. Specification, Distributed Management Task Force, June 1999.
5. Specification for CIM Operations over HTTP, Version 1.1. Specification, Distributed Management Task Force, 2002.
6. Specification for the Representation of CIM in XML Version 2.0. Specification, Distributed Management Task Force, 1999.
7. M. Debusmann, M. Schmidt, and R. Kröger. Generic Performance Instrumentation of EJB Applications for Service Level Management. In R. Stadler and M. Ulema, editors, *Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS '02)*, Florence, Italy, April 2002. IEEE Press.
8. O. Festor and A. Pras, editors. *Proceedings of the 12th IFIP/IEEE International Workshop on Distributed Systems: Operations & Management (DSOM'2001)*, Nancy, France, October 2001. INRIA Press.
9. R. Hauck. Architecture for Automated Management Instrumentation for Component Based Applications. In Festor and Pras [8], pages 231–242.
10. A. Keller, H. Kreger, and K. Schopmeyer. Towards a CIM Schema for RunTime Application Management. In Festor and Pras [8], pages 217–230.