IBM Research Report

An Information Model for Metering and Accounting

Vikas Agarwal, Neeran Karnik, Arun Kumar

IBM Research Division
IBM India Research Lab
Block I, I.I.T. Campus, Hauz Khas
New Delhi - 110016. India.

IBM Research Division
Almaden - Austin - Beijing - Delhi - Haifa - T.J. Watson - Tokyo - Zurich

An Information Model for Metering and Accounting

Vikas Agarwal, Neeran Karnik, Arun Kumar
{avikas, kneeran, kkarun}@in.ibm.com
IBM India Research Laboratory, New Delhi, INDIA

Abstract

Various service provider models have evolved in the IT industry during the last few years. Service providers build and manage infrastructure for supplying value-added services to their customers that vary from managing their basic IT operations, to running complex business processes. The complexity of the hardware and software systems deployed in service provider infrastructure requires efficient management tools and automation. These tools assist the administrators in tasks such as resource provisioning, capacity planning, charging, fault detection etc. Automation enables the infrastructure to be 'self-managed'. Moreover, being the owners of the infrastructure, the service providers have access to huge amounts of management data that they could analyze and use to their advantage resulting in better managed systems and tremendous cost savings. Services offered by service providers require usage metering and accounting so that the consumer can be charged. It may also be required for studying usage patterns, resource provisioning, quota allocations, etc. In this paper, we present an information model that captures critical management information needed by applications to enable usage accounting of service provider's resources. We propose extensions to DMTF Common Information Model's Metrics schema and introduce a new schema for metering and accounting.

Keywords: Common Information Model, metering, accounting, metrics, schema, web services, grid services, resources, e-commerce.

1 Introduction

Recent trends in the IT industry have resulted in increased acceptance and use of the outsourcing business model. Service providers build and maintain IT infrastructures and offer them for a fee to various subscribers. The services offered could vary from low-end hardware rentals such as renting compute power, storage, bandwidth etc. to high-end complex business processes modeled as web services [5]. Drawing upon economies of scale, service providers are able to offer better, cheaper and more reliable service to their customers while keeping them abreast of rapid advancements in technology.

From a management point of view, service provider infrastructures are capable of generating huge amounts of raw data that needs to be captured, analysed and acted upon for a specific business goal. The goals may vary from fraud detection, load balancing to service level agreements management. Usage accounting is one such goal that requires generation of relevant resource monitoring data, obtaining metering information from it and applying it to update consumer accounts. Metering and accounting of usage becomes an essential component of the service provider's management infrastructure, to enable automated decision making that relies upon such information. It includes purposes such as capacity planning, charging customers, resource provisioning, enforcing site-specific policies such as usage quotas, and analysis of usage patterns.

Metering of usage can either be resource-based or service-based [1]. Resource-based metering applies to scenarios where the service providers offer their clients the use of a resource – either by *allocating* a fraction of the resource *capacity*, or by allowing the client to *consume* a certain *amount* of the resource. A compute service, file storage service, or a remote interface to a printer are examples. When capacity is allocated, the metering is done in terms of *resource usage over time* – e.g. a file storage service accounts for megabyte-hours of storage. When resources are consumed, the metering is done on the basis of the amount of resource used – e.g. a printer service may track the sheets of paper printed and the amount of ink consumed; a compute service meters its users for CPU cycles consumed. Resource usage metering scenarios most commonly occur in environments such as grid computing [4].

Service-based metering is applicable in situations where a user requests certain *function* from the service, not access to a resource. Consider an email delivery service. A user sends it an email message, addressing information, etc. and requests message delivery. This service would typically meter on the basis of the number of messages (or bytes) delivered – not the CPU time, disk space or network bandwidth consumed by the service in delivering the messages. Usage is thus expressed using *application-level* parameters rather than *server-side* resource usage metrics. Web services [5] applications typically require such service-based metering.

Problem Definition

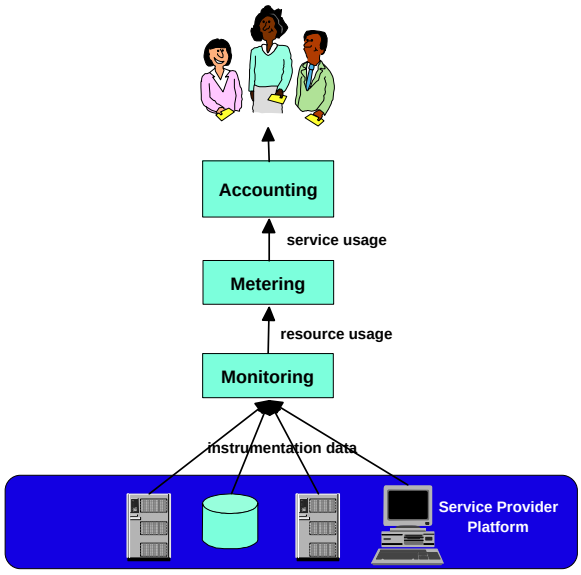

Figure 1: Information Flow for Usage Accounting

Figure 1 depicts the runtime infrastructure of a service provider, and shows the information that must flow to enable accounting. The service provider's resources, modeled as *managed elements*, are usually instrumented to generate various data items. The *Monitoring* module collects this raw data and provides it to the *Metering* module. The metering component selects the metrics that are relevant for metering the usage of the *managed element*, and processes those to compute *service usage* metrics. An *Accounting* module aggregates the usage by specific users (or accounts). Such management information needs to be modeled efficiently in order to enable applications to make effective use of it. This paper describes a model for representing usage metering and accounting information. In creating this model, our principal goals were as follows:

- Model the monitoring information generated for a unit of work.
- Model the metering information needed for the business goal of the end application. This requires defining the relationship to monitoring components from which it is derived.
- Model accounting information and its relation to the information in the metering and monitoring modules.
- Maintain compatibility with existing, related information models.

We use DMTF's Common Information Model (CIM) [3] for this purpose. CIM uses object-oriented modeling for managing systems, software, users, networks etc. Management applications can query and utilize information modeled through CIM to achieve various goals such as runtime application management [7].

Specifically, CIM Metrics schema [8] defines the notion of a *UnitOfWork* and models the representation of metrics that could be used to capture state information. In this paper, we leverage upon the CIM Metrics schema, propose extensions to it, and propose a new CIM schema to address information modeling issues that arise while representing metering and accounting information. In Section 2 we give an overview of CIM Metrics schema and discuss the existing classes and their associations. In Section 3 we describe the proposed extension to the CIM Metrics schema to support *composite metrics* and *monitoring records*. In Section 4 we describe the Metering and Accounting schema. We conclude with a summary of our work.

2 Overview of CIM Metrics schema

The CIM management model consists of a large number of classes that capture the characteristics of various system entities. It is divided into three conceptual layers:

- The *Core* model is generic (applicable to all domains).
- The *Common* models represent management information specific to a domain, but independent of any specific technology.
- The *Extension* models capture technology-specific aspects of management information.

The CIM Metrics schema[9] (see Fig. 2) is one of the CIM Common models. Its goal is twofold: First, it models a *UnitOfWork* class that is used to represent measured time taken by a unit of work. Second, it models measurements and metrics associated with managed entities such as applications, devices, systems, networks, services, etc. These metrics can also be associated with a unit of work to provide additional information about it.

UnitOfWork represents an instance of transaction or work that is either in progress or has already completed. It holds the response time and execution status of the work. It has a corresponding *UnitOfWorkDefinition*, which gives additional information about the unit of work. The unit of work instance can also be associated with the logical element (e.g., computer system, service, etc.) that executes this work, through *LogicalElementPerformsUoW*. Various types of work units can be represented using the *UnitOfWorkDefinition* and *UnitOfWork* classes. For instance, for a compute server, a unit of work may be delimited by the start and end of execution of a job. For a software service, a method invocation and its response could delimit a unit of work.

A unit of work can be composed of several smaller units of work. This is represented by *SubUoW* which correlates instances of unit of work into a hierarchy. This can be used to model nested transactions, for example. If the hierarchy is known before the execution of the transaction then *SubUoWDef* can be used to express the anticipated relationships between the instances by associating the corresponding unit of work definitions.

BaseMetricDefinition defines the properties of a metric, such as its unit of measurement, data type, etc. It is used to represent metrics such as CPU utilization, memory consumption, etc. The actual value of the metric is modeled by *BaseMetricValue*. Various values of a metric at different points of time are represented using instances of the *BaseMetricValue* class, and all are associated with the same definition via *MetricInstance*. The metric definition helps abstract out the common properties of a metric into a separate class, so that they are not repeated in every instance. These metrics and their definitions can be associated with entities like computer system, service, application, etc. by *MetricForME* and *MetricDefForME* associations respectively. These associations can be used to model information like uptime of a computer system by associating metric "uptime" and its values with the managed element "computer system".

3 Extensions to CIM Metrics schema

The CIM Metrics schema was originally designed to model application performance by representing response time for a unit of work. However the designers later increased its scope, from a single metric – response time – to capture other metrics related to a unit of work. Further, the representation of metrics and their

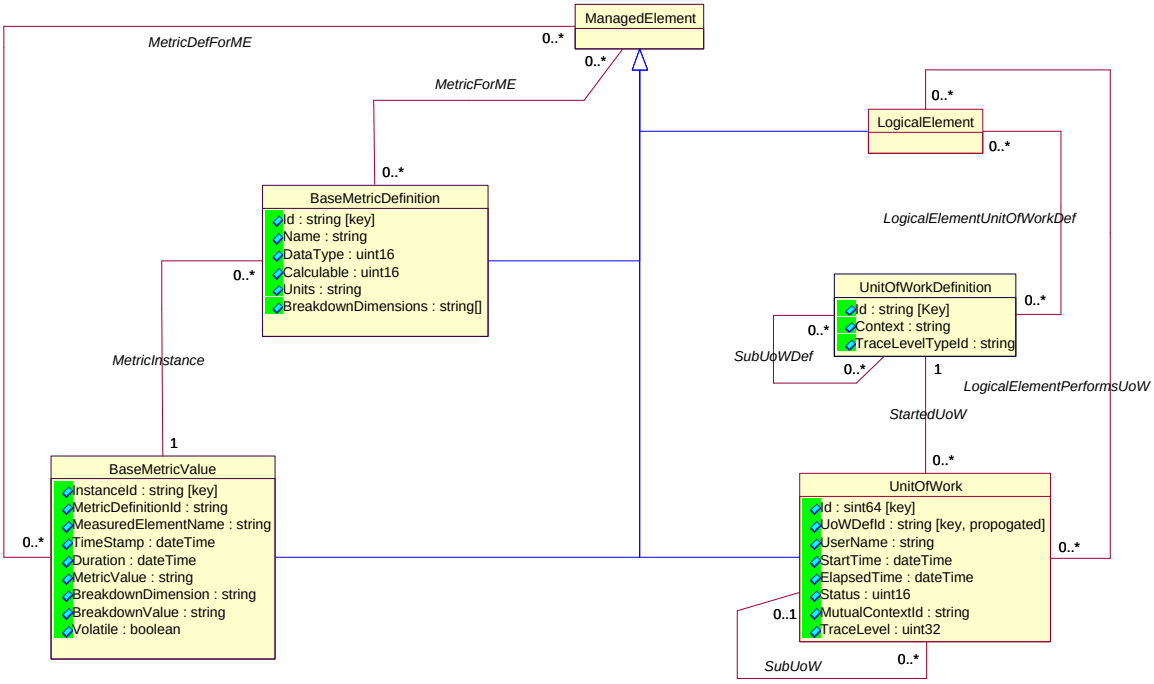

Figure 2: Relevant classes of CIM Metrics Schema

instances was decoupled from the unit of work, to allow management systems to specify metrics that were independent of any unit of work but related to a managed element. This resulted in the introduction of *BaseMetricDefinition* and *BaseMetricValue* classes. In this section we describe some concepts that are not captured by the current CIM Metrics schema, but are desirable for efficient modeling.

3.1 Introducing Composite Metrics

As described in Section 2, *BaseMetricDefinition* and *BaseMetricValue* can be used to capture raw data about the state of a managed element. *BaseMetricDefinition* is associated with a *ManagedElement* via the *MetricDefForME* association. At times it may be necessary to represent a metric that is composed of multiple base metrics. For instance, it may be desirable to represent *ResourceConsumption*, that is computed from values of two base metrics, namely, *CPUCyclesConsumed* and *AvgMemUsed*, observed within a specific period of time. The CIM metrics schema specifies that *BaseMetricDefinition* can be used to define representation of measured values as well as composite metrics such as *AvgMemUsed*. However, currently it is not feasible to represent the relationship of a composite metric (which may not be defined by the managed element but computed outside of it) with its component base metrics. This may be useful in scenarios such as auditing in commercial web/grid services where it might be important to represent how some metrics were computed. Other scenarios where composite metric representation may help include fault tolerance, fraud and intrusion detection, studying usage patterns etc.

We introduce a new class *CompositeMetricDef* (see Fig. 3), derived from *BaseMetricDefinition*, to model such composite metrics. Since a composite metric consists of several base metrics, this fact has to be captured in its definition. Therefore, *CompositeMetricDef* is associated with the corresponding *BaseMetricDefinition* classes via the *CompositionDef* association. The *CompositeMetricDef* class has a member field which describes the aggregation function required to create a composite metric from the base metrics. An aggregation function could use filtering, summation, averaging or any other function to combine the metric values. Keller et al. [6] use a *MetricAlgorithm* class that represents an algorithm, which traverses *InParameter* association to obtain input *Metric* parameters. The computation results in a *CompositeMetric* via an *OutParameter* association. We choose not to model a function in our schema because it allows only some simple functions

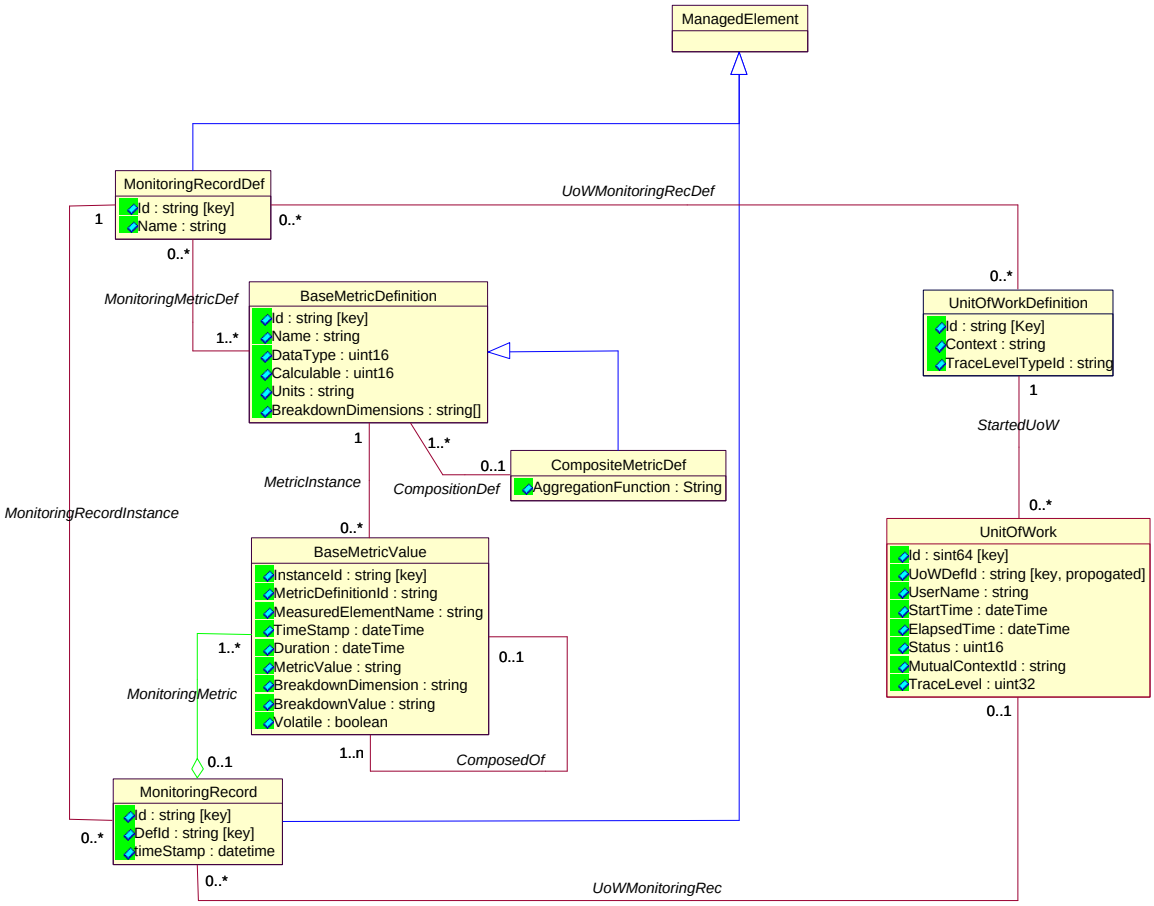

Figure 3: Modified CIM Metric Schema

to be specified. While it is possible to specify input variables as associated metrics, it is not feasible to give weights to the values of those metrics, for example. Nor is it easy to represent non-linear functions. Also, we define *CompositeMetricDef* as a subclass of *BaseMetricDefinition* since composite metric and raw metric do not seem to differ much in their representation, apart from the aggregation function that is needed with a composite metric. The instances of a composite metric are represented in our schema using the *BaseMetricValue* class. The *ComposedOf* association captures the fact that a composite metric may be computed from multiple base metrics.

3.2 Introducing a Monitoring Record

Monitoring activity is quite fundamental to management systems. It involves collection of meaningful raw data that represents the state of the entity being monitored. CIM Metrics schema allows this data to be captured through its *BaseMetricDefinition* and *BaseMetricValue* classes. However, a *BaseMetricValue* can at best represent a particular aspect of the overall state. To enable capturing of state, a representation is required that can encapsulate these individual pieces of raw state data and make the relevant state available as a single entity. For example consider *total accounting records* logged by Unix systems for every user, which contain the consumption of various resources during a day. Here, it makes more sense to represent all the metrics (such as CPU, memory and disk usage) together in a record for a user and assign it a timestamp, instead of representing them separately and associating them with a user. This can be modeled with the notion of a monitoring record — an aggregation of one or more metrics (either base metrics or composite metrics) such that the component metrics represent state observed within the boundaries of a specific period

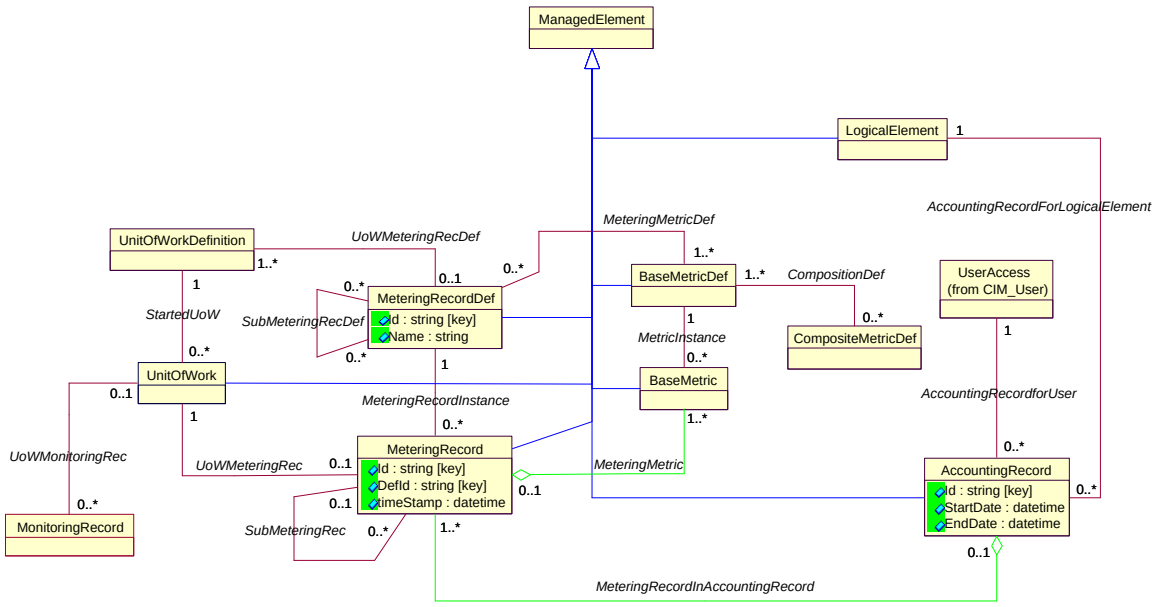

Figure 4: Metering and Accounting Schema

of time. *MonitoringRecordDef* class, defined in the context of a *UnitOfWorkDef*, is introduced to express definition of a monitoring record. Each instance of a monitoring record is represented as a *MonitoringRecord* and exists for a *UnitOfWork* specified via *MonitoringRecordForUoW* association.

A monitoring record is related to its definition through the *MonitoringRecordInstance* association. A monitoring record definition is also associated with the definitions of its component base metrics via the *MonitoringMetricDef* association. Similarly, the *MonitoringMetric* aggregation relates a monitoring record instance to its component metric instances.

We deliberately place monitoring record-related classes and associations in the Metrics schema as opposed to any other current or future CIM schema. The main reason behind this was the original intention of the CIM Metrics schema designers to model distributed application performance. Capturing the state of a managed element or unit of work at various times is as essential as capturing metrics such as response time. Currently, the Metrics schema leave it to the management application to correlate the desired metrics from a large number of instances based upon whatever criteria it chooses. Definition of a monitoring record brings more structure to that approach which is generic and is applicable to other goals such as capacity planning, fault monitoring, fraud detection etc.

4 CIM Metering and Accounting schema

The modified Metrics schema forms the foundation of the proposed Metering and Accounting schema (refer Figure 4). It is assumed that a unit of work is the entity for which metering needs to be done. A preliminary version of the proposed schema was presented at GGF5[1] [2].

As shown in Figure 4 the metering and accounting schema introduces three new classes and various associations between them and existing classes. Metering information is captured in a *MeteringRecord* that represents the information needed to measure the usage of a unit of work. The metering information is an aggregation of metering-specific metrics (base or composite) that are derived from the values contained in the monitoring records of the *UnitOfWork* under consideration. A metering record is related to its component metrics via the *MeteringMetric* aggregation association. A *MeteringRecordDef* describes the contents of a metering record. The component metric definitions are captured through the *MeteringMetricDef* association. *MeteringRecordDef* allows multiple types of metering records to be defined which might vary for different

[1] GGF : Global Grid Forum. Visit http://www.gridforum.org .

*UnitOfWork*s. In the absence of this element, an abstract *MeteringRecord* class would have to be defined and each time a new metering record has to be introduced for some *LogicalElement*, a new subclass of the abstract class would have to be created.

Each *MeteringRecord* is associated with its definition via the *MeteringRecordInstance* association. A unit of work definition is associated with its metering record definition via the *UoWMeteringRecDef* association. Similarly, a unit of work instance is associated with its corresponding metering records via the UoWMeteringRec association.

A metering record may be associated with other metering record instances via the *SubMeteringRec* association. This is because the CIM Metrics model includes composite *UnitOfWork*s and the metering record of a composite *UnitOfWork* would be computed from the metering records of its component *UnitOfWork*s. The use of *SubMeteringRec* association enables the correlation of instances of metering records. Similarly, the metering record definition of a composite *UnitOfWorkDef* is also associated with metering record definitions of component *UnitOfWorkDef*s via the *SubMeteringRecDef* association. If *UnitOfWork* hierarchies are predefined then *SubMeteringRecDef* can be used to express the anticipated relationship between the corresponding *MeteringRecDef*s. In this way, the model allows precise specification of what constitutes the metering record of a composite *UnitOfWork*. It is important to observe that *MeteringRecordDef* of a *UnitOfWork* (composite or otherwise), once defined (i.e. instantiated), does not change irrespective of which *SubUnitOfWork*s finally compose to form the composite *UnitOfWork*.

Unlike metering which is tied to a unit of work, accounting involves two participants - a producer and a consumer. In our schema, a *LogicalElement* represents the producer and a user, who consumed that LogicalElement, is the consumer. The producer could be a resource that is consumed or a service that offers something to its users and both of these are *LogicalElements* in CIM. A user is represented by *UserAccess* from CIM User schema. The consumption or use of a *LogicalElement* is modeled by a unit of work. Therefore, the usage resulting from the execution of a unit of work needs to be accounted for by the user who is responsible for initiating it.

We introduce an *AccountingRecord* to represent the accounting information that needs to be captured. It is defined as an aggregation of metering records corresponding to one or more *UnitOfWork*s which were executed on behalf of a user. An *AccountingRecord* exists in the context of a *LogicalElement* and a user of that logical element. The user could be a person, or another *LogicalElement* such as an application or service. An *AccountingRecord* is associated with its component metering records via the *MeteringRecordInAccountingRecord* aggregation. The *AccountingRecordForLogicalElement* association relates multiple instances of *AccountingRecord* to an instance of *LogicalElement*. The semantics of this association specifies to which instance of *LogicalElement* do those *AccountingRecord*s belong. Similarly, all the accounting records are associated with their corresponding users via the *AccountingRecordForUser* association.

Unlike an accounting record, the proposed schema defines a *MeteringRecord* in the context of a *UnitOfWork* and not for a *LogicalElement* or a *ManagedElement* because it is reasonable to assume that something that is meterable should have a *start-time* and an *end-time* (otherwise the metrics obtained represent monitoring information rather than metering information). Thus, anything that has to be metered can be expressed as a *UnitOfWork*. A *UnitOfWork*, in turn, is associated with a *LogicalElement* and therefore metering of an activity of a *LogicalElement* can be represented in the schema.

5 Summary

Monitoring and usage metering of system infrastructure are fundamental requirements for system management. We identified the various types of information that need to flow in a management system, in order to enable accounting of resource or service usage. We used the Common Information Model to represent this information. For this purpose, we built upon the CIM Metrics model, extending it to incorporate composite metrics and monitoring records. We then defined our Metering and Accounting schema, associating its entities with those in earlier CIM schemas where appropriate. The information gathered in this fashion has multiple potential consumers. It could be used by billing systems to generate bills for usage, by capacity planning systems for resource provisioning, by auditing systems for verifying service-level agreements, etc. We are now working on applying the Metering and Accounting schema in the grid computing scenario, involving the shared use of computing and other resources by multiple users on a grid.

References

[1] Vikas Agarwal, Neeran Karnik, and Arun Kumar. Metering and Accounting for Composite e-Services. In *Proceedings of First IEEE International Conference on Electronic Commerce (CEC 2003), Newport Beach, California (to appear).*, June 2003. An extended version is available as IBM Research Report No. RI02024 at http://domino.watson.ibm.com/library/cyberdig.nsf/Home.

[2] Vikas Agarwal, Neeran Karnik, Arun Kumar, Sugata Ghosal, Bill Horn, and Andreas Maier. Open Grid Services Architecture and Metering Resource Usage. BOF session, Resource Usage Service WG, GGF5, Global Grid Forum, Edinburgh, Scotland, July 2002.

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

[4] Ian Foster, Carl Kesselman, Jeffrey M. Nick, and Steven Tuecke. Grid services for distributed system integration. *IEEE Computer*, 35(6):37–46, June 2002.

[5] S. Graham, S. Simeonov, T. Boubez, G. Daniels, D. Davis, Y. Nakamura, and R. Neyama. *Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI.* Sams, 2001.

[6] Alexander Keller, Gautam Kar, Heiko Ludwig, Asit Dan, and Joseph L. Hellerstein. Managing Dynamic Services: A Contract Based Approach to a Conceptual Architecture. In *Proceedings of the 8th IFIP/IEEE Network Operations and Management Symposium (NOMS), Florence, Italy*, April 2002.

[7] Alexander Keller, Heather Kreger, and Karl Schopmeyer. Towards a CIM Schema for Runtime Application Management. In *Proceedings of the 12th IFIP/IEEE International Workshop on Distributed Systems: Operations & Management (DSOM), Nancy, France*, October 2001.

[8] Common Information Model (CIM) Metrics Model, Version 2.6. Distributed Management Task Force, http://www.dmtf.org/standards/documents/CIM/DSP0141.pdf, May 2002.

[9] CIM Metrics Schema, Version 2.7. Distributed Management Task Force, http://www.dmtf.org/standards/documents/CIM/CIM_Schema27/CIM_Metrics27.pdf, August 2000.