Autonomous Systems Management
Using the
Common Information Model

Honours Project Report
COMP 4905
Carleton University

Supervisor:
Dr. Anthony White
School of Computer Science

Submitted By:
Dan Byers (100280957)

Email:
dan@danbyers.ca

Friday, April 15th, 2004.
Acknowledgements

Heartfelt thanks go to Professor White. You have provided limitless technical advice, professionalism, and experience to allow me to carry through and complete this project.

To Crystal, without your endless support and encouragement, this project would not exist.

To Mom and Dad, through teaching me dedication, perseverance, and how to work hard, you have allowed me to accomplish what is no longer considered the impossible.
# Table of Contents

1.0 Problem and Background Information ........................................... 6  
  1.1 The Quality of Service Economy ............................................ 6  
  1.2 The Evolution of Service ..................................................... 6  
    1.2.1 Low-Level Redundancy ............................................... 6  
    1.2.2 Management Systems ................................................ 7  
  1.3 Traditional Management Abhors Diversity ................................ 8  
  1.4 The Attempt to Standardize ................................................. 8  
  1.5 Next Generation Management ............................................. 9  
  1.6 Intelligent Management ................................................... 10  
  1.7 Business Processes ....................................................... 11  
  1.8 Motivation ................................................................ 11  
2.0 Design of the SMS Management System ..................................... 12  
  2.1 System Requirements .......................................................... 12  
  2.2 System Architecture .......................................................... 12  
  2.3 Service Level Agreements (SLAs) .......................................... 14  
  2.4 SLA Representation in the CIM ........................................... 14  
  2.5 Action on Dependency Relations in the CIM ......................... 15  
  2.6 Providers .................................................................... 16  
3.0 Development ............................................................................ 18  
  3.1 Goals ........................................................................ 18  
  3.2 Implementation with the CIM .............................................. 18  
  3.3 Management Service Implementation .................................... 20  
  3.4 Deployment Service Implementation ..................................... 20  
  3.5 Measurement Service Implementation ................................... 21  
  3.6 Condition Evaluation Service (CES) Implementation ............ 22  
  3.7 Web Service Implementation ............................................. 23  
  3.8 System Collaboration ....................................................... 24  
4.0 Results .................................................................................... 27  
  4.1 Problems Encountered and Lessons Learned ....................... 27  
  4.2 Alternative Technologies .................................................... 28  
  4.3 Future Work and Improvements ......................................... 28  
  4.4 Conclusion ................................................................ 30  
5.0 References ............................................................................ 31
**Table of Contents, continued**

Appendix A – Software & Technologies Used in the SMS ............................. 32
Appendix B – Installation Instructions ...................................................... 33
  B-1 Installing the Java SDK & Eclipse IDE ........................................... 33
  B-2 Installing and setting up the Apache HTTP Server ......................... 33
  B-3 Installing and setting up Apache Tomcat ...................................... 34
  B-4 Installing and setting up Apache Axis .......................................... 35
  B-5 Installing and setting up Drools ................................................. 36
  B-6 Installing and setting up WBEMServices ................................... 37
  B-7 Setting up the SMS Management System .................................... 37
  B-8 Using the System ........................................................................ 38
Appendix C – Diagrams ....................................................................... 41
Appendix D – CIM Schema Extensions and Providers .......................... 56
List of Figures

Figure 1: Class Diagram #1 for the provider package ........................................... 41
Figure 2: Class Diagram #2 for the provider package ........................................... 42
Figure 3: Class Diagram for SLA CIM Schema Extensions ................................. 43
Figure 4: Class Diagram for Action on Dependency CIM
          Schema Extensions .................................................................................. 44
Figure 5: Class Diagram for Management Service Relations ............................. 45
Figure 6: Class Diagram for Condition Evaluation Service Relations ............... 46
Figure 7: Class Diagram for Measurement Service Relations ............................. 47
Figure 8: Class Diagram for Web Service Relations .......................................... 48
Figure 9: Collaboration Diagram for System Startup ........................................ 49
Figure 10: Collaboration Diagram for New Web Node Startup .......................... 50
Figure 11: Collaboration Diagram for New Proxy Startup .................................. 51
Figure 12: Collaboration Diagram for Retrieving a New Metric
          From a Registered Web Node ................................................................... 52
Figure 13: Collaboration Diagram for Updating a Monitored Service .............. 53
Figure 14: Collaboration Diagram for a Web Client
          Connecting to a Proxy ............................................................................... 54
Figure 15: System Architecture .......................................................................... 55
1.0 Problem and Background Information

1.1 The Quality of Service Economy

In order to compete in today’s global marketplace, businesses are required to be more agile and more cost-efficient than ever before. Such competition forces businesses to have guaranteed services twenty-four hours per day, seven days per week, 365 days per year. Achieving these goals is not a trivial matter.

Technology’s pervasiveness has brought itself into every aspect of the corporation: from delivering email and instant messaging, to carrying millions of simultaneous telephone conversations. The systems required to run such services are expected to be incredibly hardened and free from any possibility of fault. Unfortunately, with the nature of our infinitely malleable software and increasingly complicated hardware, components can fail, services can become unavailable, and conversations, purchases, contracts, and lives can be lost. As such, the services that every member of society has come to rely upon must have the lowest amount of downtime possible.

One of the greatest difficulties in delivering on these necessary qualities of service is due to the cost of the man-power required. Businesses require more and more functionality and flexibility to be available from the software systems they use. Unfortunately, greater functionality, larger numbers of interacting components, and further integration of technology into core services does little to decrease the system’s complexity and the severity of the consequences of failure. The best way to deal with the problem is to ensure all systems are constantly monitored to ensure the demanded service levels are met.

Systems management has, traditionally, been handled by any number of systems administrators or data centre technicians. A hard drive might fail in an e-commerce server, or a web server process might fail and have to be restarted, and all problems would have to be rectified immediately by an administrator to have full service restored.

1.2 The Evolution of Service

Fortunately, over the years, a great deal of progress has been made to ensure that these issues can be corrected as quickly as possible.

1.2.1 Low-Level Redundancy

From a low-level perspective of computer systems, redundancy is one of the main methods of ensuring a service is always available. A couple of examples follow.
1) RAID (Redundant Array of Inexpensive Disks) subsystems in servers can ensure that data stored on the disk will be properly replicated to a set of others such that no data would be lost in the event of a single hard drive failure.

2) E-commerce websites with thousands of transactions occurring every minute can cost a company a catastrophic amount of money with even the smallest amount of downtime. Handling each transaction is a typical web server. It will accept the query from the visitor of the website, perform any needed calculations or communication with other back-end facilities (i.e. databases), and return the result to the visitor. However, there is one very special characteristic of this web service. The particular web server in question is but one part of a cluster of available web servers. Each web server is identical in terms of the data it manipulates and services it offers and uses. That is the critical aspect of the cluster: there are many nodes, with each identical in function. Therefore, if any one node of the cluster fails, for any reason, the other members of the cluster will still be available to complete the remainder of the transactions. If it is a very busy website, the missing server from the cluster may degrade the overall efficiency of the service, but it will still be available to carry out the mission-critical transactions. The offline node can then be serviced in due time to be placed back in production again.

### 1.2.2 Management Systems

Management systems have the utility of querying particular metrics from a monitored service, and the ability to manage basic functionality of the service. Data is collected and any needed calculation or aggregation functions can be applied to the data. The appropriate details can then be relayed back to the user in a readable format. Applications of such systems are many.

1) Bandwidth monitoring in a network can be a very critical facility. A system can be deployed that will query each of the key network devices, border routers, switches, or hubs to determine the amount of data being transmitted through the network. The data can then be compared with normal usage levels. Being able to determine whether excessive bandwidth is being used up can help determine if a particular system has been compromised, if a user is abusing the company network, or if greater bandwidth will need to be allocated as a corporation grows.

2) Automated correction of services can help ensure the highest levels of system uptime. A monitoring system can be used to query the status of a particular service, i.e. a web application server. If the application server is ever unresponsive, or is slow to respond, the service can be modified or restarted to restore the service to full functionality.
1.3 Traditional Management Abhors Diversity

One aspect of a network or computing system’s complexity is the heterogeneous nature of the many sub-systems. Building a system – a network, or an e-commerce solution – would entail the use of software and/or hardware from any number of vendors. This is wonderful as wealth is spread generously amongst the many players in the field. Additionally, all of the pieces were engineered to integrate well together to deliver the best system possible. The next step is to ensure maximum uptime for the service. Normally, this would require the system administrators to employ some form of monitoring solution. The monitoring system will be configured to query the attributes that must be monitored from each system. In this aspect, the components were not very well engineered. Each component can have its management data represented in a different way, using its own semantics, data structures, and protocols. This is a serious problem as it becomes more difficult to handle the combination all of the different types of management data into a consistent representation.

Another problem is with the ability to efficiently obtain the required statistics from the device. For example, if the administrator must ensure that all servers in an e-commerce cluster have CPU usage below a certain threshold, a method would have to be developed such that the monitoring system will automatically log into the server and query the usage statistics from the local operating system. This is neither an easy nor an efficient task, especially when there are potentially hundreds of servers to monitor.

1.4 The Attempt to Standardize

Many solutions have been used throughout the years to try to alleviate these difficulties. Standards play a very important role in management. Without standards there can be no integration of services, no flexibility, and no end-to-end interoperability.

One such solution was the introduction of the Simple Network Management Protocol (SNMP) by the Internet Engineering Task Force (IETF). It was introduced as a standard to provide all hardware and software vendors a set of “guidelines” for how to design and produce their systems to support a particular interface. Once implemented, any such interfaces can be queried to collect available statistics.

As an example, networking equipment will support an SNMP interface to query packet counts, or byte counts that have been transmitted through a network interface. Operating systems, such as Sun Microsystems’ Solaris, have included SNMP support to allow statistics to be queried on any of the system’s components and services – i.e. system uptime, CPU, and physical memory usage.

These standard interfaces allow management applications to collect measurements in a much more generic, fairly efficient, and easy to maintain way. This, in turn, offers the administrator a very effective single point of access for monitoring all services.
Even though protocols such as SNMP exist, they are only a single part of the enabling infrastructure for management systems. For systems administrators to effectively manage burgeoning infrastructures, there must be some very solid implementations on both the agent, and management sides. Management systems need to be taken to the next level of interaction with their environment.

1.5 Next Generation Management

Considering the aforementioned service-oriented businesses, systems management must now take into consideration the “business rules” or the policies that are in place. All policies must be constantly followed to ensure that any and all services operate within their specified levels. As complexity increases in a system, it also becomes more difficult to manually enforce all of these requirements. The requirement is that management systems, themselves, have the ability to interpret and act upon said business processes, policies, and services.

For a management system to be truly effective, it must have the following important high-level abilities (DMTF Technical Committee²):

1) Consist of a mechanism to collect the data from the networking and computing environments.
2) Interpret and operate based on the described business processes and services.
3) Manage the relationships between the environment’s monitored systems.

If such a management system can properly encapsulate these three simple ideas, the systems administrator’s jobs will be a great deal easier. Less time can be spent on analyzing mountains of improperly represented data, and more time can be spent on solving problems and improving the infrastructure.

This has been the goal of an organization known as the Distributed Management Task Force (DMTF). Their primary focus has been to develop a method of representing system elements, and their inter-relationships, in a much more consistent manner. Such a consistent information model is the building block from where a business can successfully integrate and use management data.

The culmination of the DMTF’s work has resulted in a schema based model known as the Common Information Model. Released in 1997, it is a model based on standard object-oriented concepts (DMTF Technical Committee¹). The traditional features of inheritance, associations, abstraction and encapsulation exist to be able to properly provide the greatest amount of flexibility when describing the environment, and give meaning to the data that is accumulated.

The CIM contains both a specification, and a schema. The specification describes the details required for integration with other management models, and the schema provides the model descriptions. The CIM schema is structured in a hierarchical format and also incorporates concepts from other standards, such as the IETF’s SNMP protocol, to ensure
a maximum amount of flexibility when describing the elements. The schema also consists of two primary models: the core model, which describes a common set of the basic requirements and vocabulary of a managed environment, and the common model, which further defines characteristics common to certain management areas while still remaining independent of any particular technology. Extension schemas have been introduced to represent certain technologies and platform specific details in the model. Visit the DMTF website (refer to the link provided in Appendix A) for a list of the current CIM model components.

By employing an object-oriented design, the complexity of the system can be greatly reduced by ensuring that high-level and fundamental concepts are properly “abstracted” out. Objects are able to encapsulate the required common characteristics, features, relationships, and behaviour of the monitored systems.

To provide further detail from the fundamental objects in the schema, inheritance is used to construct subclasses that consist of the right amount of detail and complexity. Relationships between objects are provided in a much more expressive fashion by directly modeling them through the use of dependency, and component associations. Methods for objects within the schema can also provide the users of the system the ability to act on the management data that has been acquired.

All of the previous features are to serve a number of goals. There is a need to provide a single method of dealing with both low and high-level details of systems and services through the use of abstractions, and relationships, to describe their roles. There is a need to offer businesses a cost-effective solution with a single “point of management” that does not need to be drastically altered after every product release or new service that is introduced. Another goal is to provide vendors the ability to extend the schema to build their own specific domain of support for their own products by using the extension schema. This will allow for easy integration of those products into a customer’s management solution.

1.6. Intelligent Management

An autonomous system is an aspect of computer science that is seeing a great deal of research. Classically, autonomous systems are known as a combination of a computational core, sensors, and a control system for stand-alone, independent operation. Intelligent robots are frequently considered within this description. They can be developed to have the ability to reason and react under certain conditions, communicate, cooperate, and evolve throughout their runtime.

As complexity increases with any sized computing environment, the ability to manually configure and deploy the system is no longer a scalable, reliable solution that is adaptive to change. From the concepts of intelligent systems, a direct correlation can help alleviate the necessity for systems to be less passive. There is a drive to enable such devices to not be explicitly operated by humans. Components of the system can be
designed to manage themselves throughout an adapting environment, detect failures or bottlenecks, and eventually remedy the problems.

As management systems can include details that describe business processes and services, they can analyze the provided data, and determine whether all requirements have been satisfied. Marrying up this analysis with an appropriate “intelligence” engine would provide the necessary behaviour to manage a system with very little human interaction.

1.7 Business Processes

One of the key features to use while incorporating the CIM schema into a management system is the ability to provide a direct mapping between the business policies, processes, and services. As businesses rely upon guaranteed availability and qualities of service, there will have to be a method of describing such details to allow the management system to interpret and actively ensure all requirements are fulfilled.

A common mechanism used to enforce these business policies is through the incorporation of a service level agreement (SLA). An SLA is normally an agreement between a supplier of a service and its consumer. It can help to define the availability of the service to the user, the performance targets of any component’s workload, and the thresholds of guaranteed performance and availability. Each of the services that are provided in the computing and networking environment can be provisioned an SLA that will be used by the management system to ensure all parameters of the service are met. An example of such an SLA for a monitored service would be that a request submitted to a web service should not take longer than thirty seconds to complete. If this requirement isn’t met, the SLA will define an appropriate action that would have greater capacity provisioned for the particular service.

SLAs are a crucial requirement for a management system to be considered intelligent. They provide the “rules” that the system will be operating under. The management service would carry out any necessary analysis to ensure those SLA parameters are met. Unfortunately, describing SLAs in the management system is not a trivial task as each of the metrics that are accumulated must be properly mapped to the proper parameter from the SLA.

1.8 Motivation

The motivation for this project can be attributed to past experience in developing, maintaining, and working with a network monitoring and management system. The primary goals were: to work with new technologies, apply the many skills obtained while at Carleton, and work under the supervision of someone with many years of experience in the industry. All of these easily made it to be one of the best experiences during my time at Carleton University.
2.0 Design of the SMS Management System

2.1 System Requirements

The overall goal of the system will be to build an autonomous management system. The system will be able to perform the following functions automatically.

1) receive and configure new managed elements for monitoring
2) collect metrics from the managed elements
3) analyze the metrics from the managed elements and perform actions if the parameters of the SLA are not satisfied

2.2 System Architecture

The architecture of the system is outlined in the following diagram. The diagram is available in a larger format (refer to Figure 15) in Appendix C.
From the diagram, the following components will have to be designed and implemented:

1) Deployment Service
2) Measurement Service
3) Condition Evaluation Service
4) Management Service
5) Web Service Manager
6) Web Service Web Node
7) Web Service Proxy
8) Web Service Client

The management system to be implemented will be designed to include SLA definitions that are submitted from an external SLA monitoring infrastructure. Such an infrastructure will have the facilities to generate the properly formed agreements that describe which are the necessary details to monitor from all of the available services. Each of the available services will be monitored and managed by the management system using the elements defined in the SLA. This external system will not be included in the overall design, but will be described here (Debusmann).

Initially, the SLAs will be defined through a negotiation between a service provider and customer. The customer will retrieve the set of available metrics from the service provider. Each metric type will then be translated into various SLA parameters. On both the client and service provider sides, Business Entities will ensure that the business goals and policies of the party are maintained within the details of the SLA. Upon successful negotiation, SLA monitoring tasks are delegated to the external monitoring services, such as the management system that will be built for this project.

From the above list of components to be included in the system, the interactions between each of the Web Service components would be represented by the aforementioned service customer and provider. A typical interaction in the Web Service would see the Client connect to the Proxy, which will delegate all further interactions to any available Web Node for processing. In terms of managing the elements in this interaction, the Web Service Client would, before using the Proxy, negotiate the details of an acceptable SLA. This SLA would then be monitored by the management system throughout the series of transactions between the Web Service Client and Proxy service.

Designed to accept these new SLAs, the Management Service will receive and submit the details to the Deployment Service. The Deployment Service will check the SLAs validity, initialize an internal representation of the details, and distribute the appropriate configuration details to the Condition Evaluation Service (CES) and Measurement Service.

The Measurement Service maintains and handles information on the current system configuration and runtime metrics. This service will retrieve metrics from the resource providers that have been registered to the services that are being monitored. Such services to be monitored will be from the available Web Service Manager, Web Nodes,
and Proxy services. After data has been retrieved and compiled by the Measurement Service, the new values will be sent to the CES for analysis. The new values will undergo evaluation based on the details of the configured SLA. If the CES determines that the SLA is being violated, it will notify the Management Service that something must be done.

Upon being notified about a SLA violation, the Management service will perform any necessary action on the associated component to properly deal with the issue.

In order to realize these capabilities, the system will incorporate the previously discussed CIM schema. The CIM schema will be extended to provide support for the previously mentioned features.

2.3 Service Level Agreements (SLAs)

The following are the elements that make up a general SLA:

- **Parties**: identifies the contractual parties and contains their technical properties
- **Service description**: specifies the characteristics of the service and its observable parameters:
  - **Service operations** have one or more bindings, and one or more SLA Parameters of the service (i.e. throughput, response time)
  - **SLA Parameter** refers to one Metric
  - **Metric** can refer to one or more (composite or raw) metrics relating to the measurement directive or a function
  - **Measurement directives** specify how an individual raw metric can be obtained from a resource, i.e. a URL, SNMP GET message
  - **Functions** are the measurement algorithm specifying how the composite metric is computed, i.e. mean, sum, minimum
  - **Schedules** are defined for each function to describe the time intervals during which the functions are executed to retrieve and compute metrics

2.4 SLA Representation in the CIM

The CIM Policy schema currently includes classes that represent policies. **CIM_Policy** aggregates a number of conditions and actions through **CIM_PolicyCondition** and **CIM_PolicyAction**. Unfortunately, the CIM Policy schema does not provide a class to explicitly specify conditions and actions. The class **CIM_PolicyTimePeriodCondition** was considered to map the concept of an SLA Schedule. This system will extend that class into **SMS_Schedule**.

Next, the schema must be extended to represent the SLA parameters and their associated metrics. The CIM Metrics schema defines **CIM_BaseMetricDefinition** and **CIM_BaseMetricValue**. The system will construct new metric definitions to represent each SLA parameter, and raw or composite metric values will be calculated based on the
definition. Raw metrics are the unprocessed values retrieved directly from the monitored resource. Composite metrics consist of one or more raw metric values that have had a particular function applied (i.e. sum, average, min, max, et cetera). All required metric definitions are initially deployed to the Measurement Service which will handle the necessary aggregation of the raw and composite metrics.

Please refer to Figure #3 in Appendix C which shows the extensions that will be made to the CIM schema to allow the details of SLAs to be properly represented. A description of the extensions is as follows.

- **SMS_SLA**: is the root class and extends CIM_ManagedElement. Multiple SLAs can be created for each monitored resource and operate based on individual instances of this class being instantiated

- **SMS_MetricDefinition**: extends the details from CIM_BaseMetricDefinition and is further sub-classed to include SMS_RawMetricDefinition, which provides the details on how to collect the SMS_RawMetric

- **SMS_Schedule**: extends the details from CIM_PolicyTimePeriodCondition, and provides further Interval details. It is subclassed to an SMS_TimeSeriesDefinition to help define the SMS_TimeSeries data in terms of the “Window” in time from which values should be collected

- **SMS_TimeSeries**: extended from SMS_Metric and CIM_RawMetricValue, is defined to assist in the calculation of composite metrics that require multiple values to calculate the necessary metric. The single SMS_TimeSeries class helps to reduce redundancy with objects by encapsulating multiple values.

These metric types help to describe any number of SLA parameters. These classes are instantiated when a new SLA is deployed to the Measurement Service. Upon their instantiation, new values can be collected from the managed resources and metric values can be computed.

### 2.5 Action on Dependency Relations in the CIM

For the Management System to fully administer the monitored service, a further extension to the CIM schema is required. When a monitored service needs to be updated, or have some action performed upon it, the Management System will need the actions to be fully described in the CIM schema. Additionally, if there are associated services that are dependent on the service being updated, properties may have to be updated or actions may have to be performed on each of the dependent services.

Essentially, the system must be able to describe, and determine, that if a certain event occurs, an appropriate action must take place. For example, if the Antecedent’s status changes, all of the dependent devices will need to be updated automatically.
Please see Figure #4 in Appendix C which shows the extensions that will be made to the CIM schema to allow the details of how actions on dependencies will be properly represented. A description of the extensions is as follows (Sibilla)

- **SMS_Dependency**: extends CIM_Dependency to include a dependency between two SMS_ManagedSystemElements.

- **SMS_DependencyAction**: extends CIM_Dependency to include a dependency between an SMS_Dependency and an SMS_ActionOnDependency instance. Upon instantiation, it will follow the dependency details and update the dependent SMS_ManagedSystemElement instances

- **SMS_ActionOnDependency**: an abstract class that will allow the specification of what event is waited upon, and what action must be performed as a result.

To specify the resulting action to perform on the dependent instance, several subclasses will be defined from the SMS_ActionOnDependency class. The dependent’s target element can be an instance, property, or method, and the various actions on which can be:

- **SMS_InstanceAction**: apply delete, lock, unlock, update, reset operation on the dependent instance
- **SMS_PropertyAction**: apply a change in the dependent’s property defined by the following sub-classes:
  - **SMS_GaugePropertyAction**: to increase/decrease the target’s counter
  - **SMS_PropertyValueMappingAction**: to copy the set of property values from the source to the target’s set
  - **SMS_AffectedPropertyAction**: set the new value to the target’s property
  - **SMS_PropertyToPropertyAction**: target’s property updated by a Dependent’s or Antecedent’s property
  - **SMS_MethodPropertyAction**: property is updated by the result of an invoked method
- **SMS_MethodAction**: apply an invoke, lock, unlock on the dependent’s method

### 2.6 Providers

A provider, associated to a particular CIM class, is an implementation that provides the instantiated class object’s required functionality. When a class is instantiated, its provider is used to manage the properties and methods that are associated to the class.

Traditionally, providers for classes only had a passive role to handle its details, without maintaining any state of an instantiated object. With the new schema extensions to represent the SLAs and actions on dependencies, the providers must now have an active role while carrying out its operations.
To implement state in CIM object instances, there are a number of different providers available.

a) Instance Provider
- handles creation/deletion of CIM object instances
- allows the instance to be accessed and provided to the CIMOM
- provides a dynamic list of instances available in a CIMOM

b) Method Provider
- required to implement a method for a class
- an example could be a shutdown sequence, or resetting the counters on a network card

c) Property Provider
- provides data from a managed resource’s property, directly from the managed resource itself
- an example could be the packet throughput from a network card

d) Associator Provider
- provides the ability to dynamically build associations between objects within the schema
- an example could be when a previously unassociated/inactive drive is not being used, and is made logical or active, the AssociatorProvider will return an association between the instances of the logical volumes and the storage volumes.

3.0 Development

3.1 Goals
The overall goals of this system were to implement a management system that will automatically configure itself, collect management data, and resolve “problems” that may occur with any of the monitored services. The design showed great promise in terms of adapting the CIM schema, but an implementation was required to provide a simple proof of concept that these goals could be achieved.

The implementation was developed in Sun Microsystems’ Java programming language and framework, and incorporated the use of a few popular technologies to achieve the primary goals.

3.2 Implementation with the CIM

The central component of the system is the CIM Object Manager (CIMOM). It acts as a registry for all of the classes defined in the CIM schema. An API is available to connect to the CIMOM and create, retrieve, modify, and delete both classes and instances, and to send events – known as indications – to other instances that exist in the manager.

The project makes use of the Java based CIMOM developed by the Solaris WBEM Services group. It is a very good implementation that completely incorporates the schema and provides the required operations for managing all CIM classes and instances, with support for custom providers.

While defining the CIM schema, the DMTF provided a mechanism to fully describe all characteristics of a class through the use of the Managed Object Format (MOF). Each class, and all of its encapsulated properties and methods, is defined using the MOF. For the required CIM extensions described in the design section, a MOF file was used to store the definitions for each of the class extensions (SMS_Main.mof can be found under the root project directory). The Solaris WBEM services package also includes a MOF compiler which can be used to parse and register all of the class details from the content of a MOF file into the CIMOM. To use the new CIM extensions, all of them were defined in a single MOF file and were compiled into the CIMOM before being used. Instructions on how to register the required extensions is available in Appendix B.

Providers were implemented for each of the main extended classes to give the necessary object functionality required throughout the lifecycle of the system. Please see Table #1 in Appendix D to view the associated providers that were implemented for each of the used CIM extensions. Also refer to Figure #1 and #2 in Appendix C to see an overview of the class structure for the providers that were implemented. Descriptions for each of the main providers are as follows.

DependencyAction_Provider
This provider handles the instantiation of an associated \texttt{SMS\_DependencyAction} object. In creating this new object, all dependent \texttt{SMS\_ManagedSystemElement} instances are to have an action performed on them. It will use the associated \texttt{SMS\_AffectedPropertyAction} details to update the \texttt{SMS\_ManagedSystemElement} instances found in the associated \texttt{SMS\_Dependency} instance.

\textbf{RawMetric\_Provider}

This provider, upon instantiation of an associated \texttt{SMS\_RawMetric} object, will obtain the associated service’s URI details from the \texttt{SMS\_RawMetricDefinition} and automatically query the service’s resource for the current value. It will then set the metric value in the instance to be later retrieved by a \texttt{SMS\_TimeSeries} instance.

\textbf{Schedule\_Provider}

This provider handles instantiation of an \texttt{SMS\_Schedule} object. Upon creation, a new \texttt{ScheduleMonitor} object will be instantiated. The \texttt{ScheduleMonitor} instance contains a Java \texttt{Timer} that will be initialized using both the Interval value from the \texttt{SMS\_Schedule} instance, and a new \texttt{ScheduleTask}. Using a Java \texttt{Timer} will ensure that this instance will handle the retrieval of new values autonomously, eliminating the need for the Measurement Service to poll for new values.

Adding the new \texttt{ScheduleTask} to the \texttt{Timer} will start the repeated execution of its \texttt{run} method. The \texttt{run} method will determine which \texttt{SMS\_RawMetricDefinition} and \texttt{SMS\_TimeSeries} instances to load. The \texttt{GetNewValue} method from the \texttt{SMS\_TimeSeries} will be invoked, and a new metric value retrieved.

The new metric value will be passed to the \texttt{evaluateMetric} method, which will submit the new value to the CES for evaluation.

\textbf{TimeSeries\_Provider}

This provider, upon instantiation of an associated \texttt{SMS\_TimeSeries} object, will have it’s \texttt{GetNewValue} method invoked. Upon invocation, the system will load the required \texttt{SMS\_RawMetricDefinition} to determine the proper \texttt{SMS\_RawMetric} instance to use to obtain a new metric value.

\textbf{3.3 Management Service Implementation}

The Management Service was implemented to act as a primary entry point for new SLAs to be initialized for new managed resources. Please refer to \textit{Figure \#5} in \textbf{Appendix C} to see an overview of the class structure of the Management Service. Consisting of a Deployment Service, it will communicate with the CES, Measurement Service, and all of
the managed services. Upon system initialization, a new Deployment Service is created to handle any future configuration of the management details for any new monitored service.

The Management Service will receive notifications from the CES that a service needs to be updated due to a SLA violation. All details will be forwarded to the Deployment Service to have the appropriate actions carried out.

In order for the CES to be able to update the Management Service, it was implemented to use Java Management Extensions (JMX). JMX allows a particular interface of an object to be represented as a Management Bean (MBean). These MBeans are then registered, or exposed, with the JMX facility. A JMX facility based on the RMI protocol was used, allowing an object on a remote host to connect and invoke the methods of the MBean. The JMX facility will then translate the remote invocations to a method call on the object being represented by the MBean. The method modifyService was exposed through this management interface to allow the CES to submit service modification requests.

### 3.4 Deployment Service Implementation

Any registration of a new monitored service will need to have all appropriate SLA details deployed to the CIMOM, Measurement Service, and CES. Please refer to Figure #5 in Appendix C for an overview of the class structure of the Deployment Service.

Upon receiving a new service to register, elements from the new CIM extensions will be used to create the service’s representation in the CIMOM. The Deployment service will perform the following instantiations and operations to fully represent the new service:

1) create a new SMSManagedSystemElement object to represent the physical details of the monitored service
2) create component associations between two SMSManagedSystemElements through the use of a new SMSComponent association
3) create dependencies between two SMSManagedSystemElements through the use of a new SMSDependency association
4) create a new SMS_SLA object to represent the new SLA
5) obtain the set of available resources, or “attributes”, from the new service, these will be the monitored “resources” of the service
6) create new SMS_RawMetricDefinition objects to represent each of the available resources as parameters of the SLA
7) for each SMS_RawMetricDefinition associate it to the SMS_SLA instance by using a new instance of SMS_SLAPerParameterDefinition
8) register the details of each SMS_RawMetricDefinition and the associated SLA details, represented as threshold details, with the CES
9) register the details of each SMS_RawMetricDefinition with the Measurement Service such that new metrics can be submitted to the service and properly manipulated
10) create a new SMS_TimeSeriesDefinition to provide the value of which “window” of time to use during the scheduled retrieval of the metric’s value 
11) create a new SMS_SampleDefinition associating the SMS_TimeSeriesDefinition with the SMS_RawMetricDefinition 
12) create a new SMS_Schedule instance to represent the new schedule used to obtain the associated metric’s value 
13) create a new SMS_SamplingPeriod association between the SMS_Schedule and the SMS_TimeSeriesDefinition to associate the new schedule instance with the metric to be monitored 
14) create a new SMS_SLAValidity association which will associate the new schedule instance to the SLA instance 

The Deployment Service was implemented using JMX to provide an efficient mechanism for each new monitored service to register with the Management System. The exposed methods, addManager, webNodeRegistered, and proxyRegistered are available through the JMX facility to allow a new Web Service Manager, WebNodes and Proxy to automatically register.

3.5 Measurement Service Implementation

The role of the Measurement Service is to manage the data retrieved from all monitored services, based on the requirements of the provided SLA. Please refer to Figure #7 in Appendix C for an overview of the Measurement Service’s class structure. Each SLA parameter representing a monitored resource is defined through the use of an SMS_RawMetricDefinition. The Deployment Service will register each of these parameter definitions with the Measurement Service to allow the service to receive and process new metric values. Ultimately, the Measurement Service would apply certain defined functions upon the raw data to generate any necessary composite data, but this implementation will remain basic and will not incorporate this type of functionality.

One of the key design issues was to avoid having the Measurement Service perform the required periodic polling of the new metric values. Solving this problem involved the instances of SMS_Schedule incorporate the necessary scheduling mechanism to automatically trigger and retrieve new values. The SMS_Schedule object, realized through its associated provider, will invoke the GetNewValue CIM method in the SMS_TimeSeries instance. This would cause the collection of a new value that would be added into the SMS_TimeSeries instance. The SMS_Schedule instance, upon receiving the new value, will submit the value to the Measurement Service.

After the required aggregation or compilation functions have been performed, the value will be submitted to the CES for further analysis of the value.

The Measurement Service was implemented using JMX to provide an efficient mechanism for new metric values to be received. The exposed methods, registerMetric and processMetric, are available through the JMX facility to allow
the Deployment Service to submit new SLA metric definitions, and all \texttt{SMS\_Schedule} instances to submit new metric values.

### 3.6 Condition Evaluation Service (CES) Implementation

This system, considered to be the “brains” of the system, is to interpret measurement data and produce outcomes based on the configured parameters of the SLA. Please refer to \textit{Figure \#6} in Appendix C for an overview of the CES’s class structure. The Deployment Service will use the details of the SLA to define the necessary thresholds, or other specific quality of service requirements, for each of the \texttt{SMS\_RawMetricDefintions} used to represent a service’s resource. Each “threshold” on a resource is then deployed to the CES by the Deployment Service.

The CES will receive new metric values from the Measurement Service. For each value received, it will compare the metrics to the associated thresholds previously received, and, based upon these rules, calculate an appropriate outcome. Since each of the SLA attributes are limits or other operations based on values, they can be expressed as rules.

Rule based systems are a fantastic facility to engender “intelligence” into a system. The underlying concepts are that rule based systems represent knowledge in terms of a number of facts, or assertions, that describe any form of beginning state of the system. A set of rules is also provided that describes how to act upon the assertion set. Upon a query being sent to the system with a certain value of input, the input will be tested against the known facts. Ultimately, an outcome can be derived depending on whether the input can produce a true, false, or even that a solution doesn’t exist.

There are two popular types of rule based systems: forward chaining, and backward chaining. A rudimentary description of each sees backward chaining systems as query based with an application querying the fact space to determine trueness. Prolog is one of the original backward chaining systems. Forward chaining systems are event based and consequences are triggered based on information in the fact space.

For this implementation, a forward chaining engine was used to implement the required behaviour of the CES. Drools is an open source forward chaining rule based system that has been adapted to the Java framework. This support for Java objects allows for better expression in terms of the required business rules associated to any business objects.

The set of rules used in the CES’s rules engine were only of a primitive form of \texttt{IF-THEN} conditions, but the ability to expand on this logic exists for future iterations. The elements of the SLA, provided as a set of thresholds, are used as “conditions” during evaluation. Each threshold is represented using the \texttt{MonitoredService} object from the CES package. Upon receiving a new metric value, such objects are asserted into the workspace of the drools engine and the associated value is then evaluated by the logic from the \texttt{MonitoredService} object. Outcomes are determined based on whether or not the SLA’s threshold was satisfied.
The design of the CES was to have it made available as a web service to allow the greatest ease in remotely querying the service. Apache Tomcat was used for the container to house the CESServlet, and JAX-RPC was used to invoke the `receiveMetric` and `setResourceThreshold` methods used to query and manage this service.

Having the CES published as a web service allows for even easier distribution of components. They can be sitting in different segments of the network without having to be concerned with details of firewalls – since firewalls are usually open to web traffic. Using a web service design also allows for easy deployment within Tomcat, or any other container, which can quickly be deployed within a web server on any architecture.

### 3.7 Web Service Implementation

In order to test the Management System implementation, it was necessary to provide a service than can be monitored. An important detail that was considered was that it should emulate something very critical in today’s high-availability infrastructure. For that reason, a standard web server was implemented. The primary feature of this web server is that, aside from its ability to handle basic requests from a “web client”, it runs as a clustered service. It incorporates a primitive proxy mechanism that evenly distributes requests to each of the available nodes. The Proxy uses a round-robin routing of requests to all of the available Web Nodes. Please refer to Figure #8 in Appendix C for an overview of the Web Service class structure.

The web server was designed to provide complete manageability of all elements. This is accomplished through the exposure of management interfaces through a JMX facility. Each of the Web Nodes, the Proxy, and the Manager all have interfaces implemented using MBeans. Each component service can be registered with the Management System which will discover and monitor any available resources from each service. The resources available on each service will be discovered through the standard Java introspection facility.

Upon the startup of the Web Service Manager, it will register itself with the Deployment Service. The Deployment Service will then ensure that the Manager service has been properly configured in the CIMOM and the rest of the Management System’s components – see the previous discussion of the Deployment Service Implementation for these details.

The Manager maintains references to, and handles all aspects of management for all Web Nodes and Proxies in the system. Ideally, if the CES has determined an SLA violation, the appropriate Manager’s operation will be invoked to remedy the problem. For example, if the service is violating request time thresholds, the CES can invoke the Manager’s `activateNode` method. This will cause the Manager to distribute a new Web Node to the proxy to provide more processing resources. For this implementation, however, the Manager only handles the registration of each of the Web Nodes and Proxy. Specific management functionality should be included in future iterations.
When either a Web Node or a Proxy is initialized, they will be registered with the Manager. Once registered, the Manager will push all inactive Web Nodes to the Proxy service. This will allow each of the Web Nodes to start receiving requests relayed through the Proxy. The Manager will also register all Proxy and Web Node services with the Deployment service. The Deployment Service will ensure that each element is added to the CIMOM and has all available resources being monitored using the proper schedule.

The management interface for the Proxy supports the ability to add a Web Node from its set of active clients, retrieve request counter values and status information. The Web Node’s management interface supports the retrieval of the status of the node, and the number of requests that it has received since initialization.

### 3.8 System Collaboration

Please review the provided figures in the Appendix that describe the primary interactions that occur within this system. Their description is as follows.

**Figure 9: Collaboration Diagram for System Startup**
- Measurement Service initializes
- CES initializes, and is, usually, constantly available through the Web container that it exists in
- Management Service initializes, starts the Deployment Service, and registers its details with the CES
- Web Service Manager initializes and registers with the Deployment Service
- The Deployment Service creates the appropriate instances in the CIMOM

**Figure 10: Collaboration Diagram for New Web Node Startup**
- Web Service Node is initialized and registers with the Manager
- If a Proxy Service exists, the new Web Node will be registered with the Proxy, if not the Web Node remains in an inactive state
- The Manager sends registration notification to the Deployment Service
- The Deployment Service performs all necessary instantiations, registers all details of the new service, schedule, and associated SLA

**Figure 11: Collaboration Diagram for New Proxy Startup**
- Web Service Proxy is initialized and registers with the Manager
- If any Web Nodes exist, all Nodes will be registered with the new Proxy Service, if not, the Proxy will wait for new Web Nodes to be registered
- The Manager sends registration notification to the Deployment Service
- The Deployment Service performs all necessary instantiations, registers all details of the new service, schedule, and associated SLA
Figure 12: Collaboration Diagram for Retrieving a New Metric From Registered Web Node

- With the Web Node’s monitored resource information configured in the CIMOM, the SMS_Schedule instance will have a new ScheduleMonitor instance running the instance of the ScheduleTask
- Each ScheduleTask is executed at the configured interval, and will instantiate and invoke the GetNewValue method from an associated SMS_TimeSeries instance
- The SMS_TimeSeries instance will obtain the new value from a new SMS_Metric
- The SMS_Metric instance will actively query the configured Web Node’s monitored resource and store the new metric values
- Once the metric value has been obtained, the SMS_Schedule instance will submit the new value to the Measurement Service
- The Measurement Service will process the metric, and submit the value to the CES for processing

Figure 13: Collaboration Diagram for Updating a Monitored Service

- Upon receiving a new metric value, the CES will retrieve the associated metric definition’s threshold details from the proper MonitoredService instance
- The MonitoredService will be used (asserted into the rules engine) and the metric value will be evaluated against the threshold
- If the threshold is violated, the Management Service will have its remote method, modifyService, invoked and it will send the details to the Deployment Service
- The Deployment Service will use the provided metric definition to determine what actions must be performed. The actions are defined using the SMS_AffectedPropertyAction object, and the service immediately referenced will be updated.
- An SMS_DependencyAction object will then be created. It will reference the SMS_AffectedPropertyAction object and any SMS_Dependency objects that are associated to the original service’s SMS_ManagedSystemElement object. Each of these dependent services will also be updated by the SMS_DependencyAction instance

Figure 14: Collaboration Diagram for a Web Client Connecting to a Proxy

- The existing Web Service elements, Manager, Proxy, and at least one Web Node will have been initialized
- A new Web Service Client will be initialized and it will automatically connect to the Proxy Service listening on a provided port
- Upon receiving the new connection, a new ProxyAccept handler will be initialized, and will connect to the next available Web Node
- Upon receiving the new connection, the Web Node will initialize a new NodeAccept handler to manage the remaining transactions, and return the response, through the Proxy, to the Client Service.

4.0 Results

4.1 Problems encountered and Lessons Learned

There were a number of lessons learned from the difficulties encountered throughout this project. The majority of the required software in this project was open source, freely available software. This is a wonderful way to expose oneself to new technologies. As well, there is normally a decent sized community of enthusiasts to deal with support requests.
At the outset of the project, it was decided that the SNIA CIMOM would be checked out to see how easy it was to integrate into the system. After a brief attempt at compiling and running the software, it was determined that there were integration issues. The biggest problem was that the CIMOM was determined to be inactive, and the SNIA had abandoned it. This implied that there would be little support if there were problems encountered along the way with the project. Thus, it was abandoned for this project.

The Web Based Enterprise Management (WBEM) Services group was investigated next. Backed by Sun Microsystems, they had a very complete implementation of the CIM schema in a Java-based CIMOM. It was not very long before the CIMOM was in operation, and the learning curve began with determining how to start to implement a management system using the CIM schema.

The first major problem encountered was with regards to having intercommunication between multiple providers. One instance was not able to invoke another instance’s method using the standard connection from the provider. After emailing the support forum, it was determined that the feature was supported. Code was posted as an example of how this was being attempted, but no solutions were recommended. After combing the source to find a potential bug, and having no luck, a very simple workaround was discovered. It entailed the provider doing a physical connection, seemingly a second time, to the CIMOM. It was incredibly frustrating to have this simple workaround after spending such large amounts of time to determine the initial problem. All in all, the proper object instances were being retrieved and their methods could be invoked.

This is a frequent problem with all types of software, bugs can exist and sometimes the only solution is an easy workaround. One must learn to think “outside of the box”.

Another related issue was to determine the best rule-based system, implemented in Java, to implement for the CES. JEOPS was kindly recommended, but, unfortunately, it followed the trend of the SNIA CIMOM. With the JEOPS package no longer being supported it was considered best to be abandoned for this project. After some brief searching, the Drools forward chaining engine was selected. It integrated seamlessly into the system, and was very easy to implement the basic evaluation service required.

After a fairly lengthy attempt at exposing the CES as a web service using SOAP, it was abandoned for the easier to implement JAX-RPC. With the rapidly changing landscape of web service frameworks, it was difficult to find a standard that suited this task best. Ultimately, the easiest seemed to provide the best solution.

The WBEM Services CIMOM is a complex application. There were many little idiosyncrasies about it. When setting the properties of a particular class, and using a non-standard datatype for the property, such as a CIMInstance reference, it was determined that you must only use the CIMObjectPath derived from the CIMInstance itself. Other CIMObjectPaths that were manually constructed would not work. Again, lack of documentation was the issue here.
When developing any software solutions under a short time frame, the more robust solutions are rarely the route to go. It is best to do some research and select the technologies that appear to have the proper feature set which can be implemented in the simplest manner. This quality for estimating difficulty is normally one that only comes from experience.

Careful attention must be paid when forming the class definitions in the MOF file. Descriptions must not contain certain non-alphanumeric characters in the class, property, or method descriptions of the classes. The XML Parser used during the system’s operation will give unknown results and certain functionality may be altered. For example, including single quotes in a few descriptions broke the ability to modify or delete CIM Classes and Instances from the Object Manager.

4.2 Alternative Technologies

The Condition Evaluation Service was implemented with the Drools engine and is accessible via its web service interface with JAX-RPC. Other options can include the service being available through pure RMI. Any other forward chaining rule engine can also be easily integrated into the system. The rule sets used were pretty primitive so it wouldn’t be a chore to include new technologies.

The Management Service will connect to the remote Web Service Manager using the exposed interface through remote JMX. Other technologies, such as JXTA, could be employed to provide a better facility to obtain remote references to distributed services.

4.3 Future Work and Improvements

As tight deadlines and problems delayed the significant amounts of progress intended, future work for this implementation could see many improvements to the system.

The Management Service could provide a web interface to have SLA details provided, or a proper facility can be developed to handle the required negotiation of new SLAs. Currently, the SLA details are hard-coded into the system.

For this implementation, composite metrics were not implemented in the extended schema. If there was composite metric support available, the SMS_RawMetric could be used to represent the raw value retrieved from the service, then aggregated into the required value and stored in the SMS_TimeSeries instance.

A nice feature to implement would be a centralized logging facility to have different levels of logging of the messages and errors that might be transmitted between components in the system.

Currently, SLAs only consist of numerical metrics to monitor. Any “state” attributes of a service are currently ignored, for example, the service’s status. A future enhancement
would be to allow an entity state’s to be monitored by the Measurement service. This would be useful if there was a clause in the SLA stating that a certain action would occur if the monitored resource becomes unavailable more than three times in a certain interval.

The Apache group has a Java application, JMeter, which can be used to stress-test any web service. This would be a good thing to implement to provide a load on the clustered web service and see how the Management System would handle other conditions.

With the CIMOM holding references to the available managed resources, it would be very beneficial to have an interface to view the hierarchical listing of available instances that are being managed. A web interface could be implemented that would provide a simple read-only view to the condition of each of the managed resources.

To allow for long term analysis on a managed resource’s metrics, it might be useful to incorporate a database solution to ensure data longevity. Historical data can then be collected and analyzed for potential trends in a service’s runtime.

Upon any of the components of the Management System failing or becoming unavailable, there will have to be measures implemented to ensure that scheduling re-commences as soon as possible within the active providers of the CIMOM once any component of the Management System becomes available again.

This implementation only dealt with schedules that only used intervals for scheduling the collection of metric values. This could be further developed to provide a more robust scheduling mechanism.

Security was not considered in this implementation. Web Services and inter-system communication was transmitted in plain text. This could be augmented for future iterations to use SSL based communication, and other related technologies. The CIMOM also includes authentication modules that were not incorporated in this implementation.

4.4 Conclusion

Given the goals of the project, it can be seen that it is possible to develop a system that exhibits the main properties listed at its outset. Unfortunately, the proof of concept was only a primitive version, but the concept is present and the potential is great for such a system.

One of the main problems with such descriptive, general platforms is that it can get “bloated” due to the need to support so many available technologies and scenarios. The CIMOM is a fairly “heavyweight” application, and the interactions that were implemented for this project entailed some fairly resource intensive operations. In order
to utilize this technology in a production environment, the system would have to be considerably refined and optimized to ensure an adequate level of scalability and robustness.

The Common Information Model holds tremendous promise in bridging the gap between the proper management of the systems available in today’s computing and networking environments. Self describing and self managing autonomous systems are definitely the future in systems management.

This project has been a very rewarding experience, and has provided a new perspective on designing and building complex systems that utilize a number of different advanced technologies.

5.0 References


Appendix A – Software & Technologies Used in the SMS

Java SDK 1.5.0
WBEM Services
  - WBEM Services v1.0.1 toolkit, API documentation, and developer’s guide
Drools Rules Engine
Apache 2.0.53
Apache Tomcat 5.5
Apache Web Services Project
Appendix B – Installation Instructions

If you do not have the CD available, the project’s package can be downloaded from Dan Byers’ website at http://www.danbyers.ca/.

B-1 Installing the Java SDK & Eclipse IDE
This management system was developed using Sun’s JDK v. 1.5.0. A copy of the SDK is provided under the /install-binaries directory.

Eclipse 3.0.2 was also used as the primary IDE. A copy has been provided under the /install-binaries directory.

It will be assumed that the reader is familiar with installing the JDK and Eclipse IDE. Please take note of the installation directory.

**B-2 Installing and setting up the Apache HTTP Server**

Locate the copy of the Apache server binary from the project package’s /install-binaries directory.

/install-binaries/apache_2.0.53-win32-x86-no_ssl.msi

Run this msi file (yes, msi extensions are executable) and follow the standard installation steps to install Apache. Make sure you remember where you installed the package as you will be referencing configuration files from within it.

The apache installation root directory will be referred to as: %ApacheRoot%.

There is a shared object module called Apache JK, which is a connector for Apache and the Jakarta Tomcat instances. This will have to be put in place within Apache.

This shared object can also be found in the /install-binaries directory

/install-binaries/mod_jk.so

This file needs to be copied into the %ApacheRoot%/modules directory. Please do so.

From the directory /apache_conf a file called workers.properties must be copied over to the %ApacheRoot%/conf directory.

In the %ApacheRoot%/conf directory, open the file httpd.conf in your favourite editor.

Browse to the end of the section that includes a number of #LoadModule definitions.

Add this text to a line below:

```
LoadModule jk_module modules/mod_jk.so
<IfModule mod_jk.c>
JkWorkersFile conf/workers.properties
JkLogFile logs/mod_jk.log
JkLogLevel info
JkLogStampFormat "[%a %b %d %H:%M:%S %Y] "
JkOptions +ForwardKeySize +ForwardURICompat -ForwardDirectories
JkRequestLogFormat "%w %V %T"
JkMount /* worker1
</IfModule>
```
When finished, save and exit the file.

**B-3 Installing and setting up Apache Tomcat**

Locate the copy of the Apache Tomcat server binary from the package’s `/install-binaries` directory.

```
/install-binaries/jakarta-tomcat-5.5.7.exe
```

Run this file and follow the standard installation steps to install Tomcat. When it asks for a port to run on, use the default (it should be 8080). Make sure you remember where you installed the package as you will be referencing configuration files from within it.

The Tomcat installation root directory will be referred to as: `%TomcatRoot%`.

Edit the `server.xml` file from the `%TomcatRoot%/conf` directory.

Browse to the section that includes the configuration of the default connector. Change any of the settings to include the following:

```
<Connector port="8080" maxThreads="150" minSpareThreads="25" maxSpareThreads="75" enableLookups="false" redirectPort="8443" acceptCount="100" connectionTimeout="20000" disableUploadTimeout="true" />
```

Browse further down the file to where the default virtual host is located. Change the settings to include the following detail:

```
<Host name="localhost" appBase="webapps" unpackWARs="true" autoDeploy="true" xmlValidation="false" xmlNamespaceAware="false">
```

That should be all that is required in terms of configuration.

Restart both of the Apache HTTP and Tomcat instances. Remember to always shutdown Tomcat first, then the HTTP Server. Then start up the Tomcat server first, then the HTTP server.

---

**NOTE:** For the following packages/services listed in the next sections to be installed, it would be easiest to pick a common directory for the install location for all of them, i.e.

```
C:\honoursProject\utils
```

This will provide a central location in which to look for all required libraries.
B-4 Installing and setting up Apache Axis

Locate the copy of the Apache Axis binary from the project package’s /install-binaries directory.

/install-binaries/ axis-bin-1_2RC3.zip

Install this package and note the installation directory. Denoted as %AxisRoot%

Axis will have to be configured in the Tomcat service.

Copy the entire directory structure under the following path

%AxisRoot%/webapps

into the %TomcatRoot%/webapps directory

Axis will then be automatically deployed to the container. You can verify that your Tomcat/HTTP/Axis instance is all working properly if you go to a web browser and type in the following URL:

http://localhost/axis/

If you do not get the main “Hello! … “ axis webpage, then there is something wrong with the installation. Please review the previous steps and ensure they are completely followed. Sometimes a simple restart of Apache HTTP and Tomcat can solve the problem.

B-5 Installing and setting up Drools

Drools will have to be compiled from the provided source.

/install-binaries/drools-2.0-beta-21-src.zip

First, obtain the maven binary

/install-binaries/maven-1.0.2.exe

Install maven, using the default installation parameters.

Unzip the drools source package into the directory that was previously recommended.
Next compile Drools. From the command line, go into the drools source directory and execute maven. It will compile and run all tests to ensure Drools is setup properly. A better description of how to use maven to install drools can be found at:

http://www.drools.org/Maven+Build+Instructions

To use the drools environment, the following libraries will have to be included in the external jars of the builder for eclipse, or the CLASSPATH runtimes:

- drools-base-2.0-beta-20.jar
- drools-core-2.0-beta-20.jar
- drools-io-2.0-beta-20.jar
- drools-smf-2.0-beta-20.jar
- drools-examples-2.0-beta-20.jar
- drools-java-2.0-beta-20.jar
- antlr-2.7.2.jar
- janino-2.0.16.jar
- xml-apis-1.0.b2.jar
- xercesImpl-2.6.2.jar

If these libraries cannot be found, they have been included in the /lib directory of the project’s package.

These libraries will have to be copied over to the Tomcat webapps axis lib directory.

Copy them to the following folder:

%TomcatRoot%\webapps\axis\WEB-INF\lib

B-6 Installing and setting up WBEMServices

To install the CIMOM and the required CIM libraries, unzip the following package from the package, into the recommended directory.

install-binaries/wbemservices-1.0.2.bin.zip

Be sure to note where the package was installed.

To run the CIMOM and CIMWorkshop, look at the batch files that were included in the project package’s directories:

/wbemservicesRunTime/bin
/wbemservicesRunTime/cimom-bin
They can be copied over to the respective place of the existing files from the installation. They will have to be modified to reflect your specific directory structure. They will be able to run both services.

**B-7 Setting up the SMS Management System**

This project’s implementation can be found under the `install-binaries` directory:

`/install-binaries/sms.zip`

Unzip the package to the recommended directory. The following will have to be done to get the system up and running.

1) **Modify the paths in each of the batch files:**

Edit the path variables in each of the following files to reflect the directory structure of the current machine. You will need to reference where each of the previously installed packages are located.

- client.bat
- insertClasses.bat
- manager.bat
- ms.bat
- proxy.bat
- registry.bat
- sms.bat
- start_cimom.bat
- webnode1.bat
- webnode2.bat

2) **Deploy the CESServlet**

From the Tomcat installation, create the following directory

```bash
%TomcatRoot%/webapps/axis/WEB-INF/classes/ces
```

From the project’s package, copy the package contents of the `ces` package over to the directory under Tomcat. The following files must be included:

- evaluator.drl
- CESServlet.class
- CESDDL_undeploy.wsdd
- CESDDL_deploy.wsdd

Next, the service will have to be deployed to Tomcat. From the command line, change from the current directory to the listed Tomcat directory.
Thus, once you are in the following directory:

%TomcatRoot%/webapps/axis/WEB-INF/classes/ces

Do a directory listing and verify that it contains the file CESDDL.wsdd. Then enter in the following command:

```java -cp %AXISCLASSPATH% org.apache.axis.client.AdminClient -lhttp://localhost/axis/services/AdminService CESDDL.wsdd```

This would register the new service and automatically generate the WSDL content for the webservice. Typing the same command, but referencing the CESDDL_undeploy.wsdd would remove the service from the Tomcat instance.

To verify the service is running, type this URL in a browser:

http://localhost/axis/services/cesService?wsdl

If you do not see this, go no further and review the previous steps to ensure they are being followed correctly.

The Management System should now be ready for execution.

**B-8 Using the System**

The following steps are required before each run through the system.

1) reload the CIMOM metabase (this isn’t required for the first run through)

From the provided /cimom/logr directory in the project’s package, copy over all of the files in the directory into the same CIMOM’s /logr directory under the directory that the wbemservices package was installed under.

The destination should be found under:

%WBEMSRoot%/cimom/logr

Make sure you delete the current CIMOM’s files from the destination /logr directory (after shutting down any running instances of the CIMOM)

Next, in a separate CMD window, and from being within the project’s root directory that contains all of the batch files, execute the following:

2) initialize the CIMOM by typing in:

    start_cimom.bat
3) from another CMD window, insert the set of SLA classes into the CIMOM:

    insertClasses.bat

    The command line should have returned, and you should be located in the main /bin directory of the wbem services package. Now run the CIMWorkshop.

    cimworkshop

    To log in just type in a bogus username and password as security is not implemented. Using this application will help you to visualize what is happening in the system.

4) from another CMD window, start up the RMI registry

    registry.bat

5) from another CMD window, start up the Measurement Service

    ms.bat

6) from another CMD window, start up the Management Service

    sms.bat

    - before proceeding, wait until it tells you that the Management Service has been successfully initialized

7) from another CMD window, start up the Web Service Manager

    manager.bat

8) from another CMD window, start up the first Web Node

    webNode1.bat

9) from another CMD window, start up a second Web Node

    webNode2.bat

10) from another CMD window, start up the Proxy

    proxy.bat

11) from another CMD window, start up the Web Service Client

    client.bat
This client can be run repetitively until the CES will update the Management Service since a threshold has been exceeded.

To stop any of these services, simply kill the processes by using the <CTRL-C> key combination.

After any given run through, there will be two files saved to the disk:

`thresholds.properties`, and `management.properties`

These files should be found from within the `windows/system32` directory. They can be cleaned up after every use.

In the root project directory, the file `ces.properties` will get written which will contain the URL to access the CESServlet. If the IP address of the host changes for any reason, between attempts to run the system, this file will need to be deleted.

To ensure that the system is operating correctly, make sure to also check the tomcat logs, found under the directory `%TomcatRoot%/logs`, is also getting written to with debugging information.

**NOTE:** if the system is ran a second time, make sure step #1 is completed.

**Appendix C – Diagrams**

**Figure 1: Class Diagram #1 for the provider package**
Figure 2: Class Diagram #2 for the provider package
Figure 3: Class Diagram for SLA CIM Schema Extensions
Figure 4: Class Diagram for Action on Dependency
CIM Schema Extensions
Figure 5: Class Diagram for Management Service Relations
Figure 6: Class Diagram for Condition Evaluation Service Relations
Figure 7: Class Diagram for Measurement Service Relations
Figure 8: Class Diagram for Web Service Relations
Figure 9: Collaboration Diagram for System Startup
Figure 10: Collaboration Diagram for New Web Node Startup
Figure 11: Collaboration Diagram for New Proxy Startup
Figure 12: Collaboration Diagram for Retrieving a New Metric From a Registered Web Node
Figure 13: Collaboration Diagram for Updating a Monitored Service
Figure 14: Collaboration Diagram for a Web Client Connecting to a Proxy
Figure 15: System Architecture
Appendix D – CIM Schema Extensions and Providers
<table>
<thead>
<tr>
<th>Class Name</th>
<th>Associated Provider</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMS_AffectedPropertyAction</td>
<td>AffectedPropertyAction_Provider</td>
</tr>
<tr>
<td>SMS_Component</td>
<td>Component_Provider</td>
</tr>
<tr>
<td>SMS_Dependency</td>
<td>Dependency_Provider</td>
</tr>
<tr>
<td>SMS_DependencyAction</td>
<td>DependencyAction_Provider</td>
</tr>
<tr>
<td>SMS_ManagedSystemElement</td>
<td>ManagedSystemElement_Provider</td>
</tr>
<tr>
<td>SMS_RawMetric</td>
<td>RawMetric_Provider</td>
</tr>
<tr>
<td>SMS_RawMetricDefinition</td>
<td>RawMetricDefinition_Provider</td>
</tr>
<tr>
<td>SMS_SampleDefinition</td>
<td>SampleDefinition_Provider</td>
</tr>
<tr>
<td>SMS_SamplingPeriod</td>
<td>SamplingPeriod_Provider</td>
</tr>
<tr>
<td>SMS_Schedule</td>
<td>Schedule_Provider</td>
</tr>
<tr>
<td>SMS_SLA</td>
<td>SLA_Provider</td>
</tr>
<tr>
<td>SMS_SLAParameterDefinition</td>
<td>SLAParameterDefinition_Provider</td>
</tr>
<tr>
<td>SMS_SLAValidity</td>
<td>SLAValidity_Provider</td>
</tr>
<tr>
<td>SMS_TimeSeries</td>
<td>TimeSeries_Provider</td>
</tr>
<tr>
<td>SMS_TimeSeriesDefinition</td>
<td>TimeSeriesDefinition_Provider</td>
</tr>
</tbody>
</table>