A Data Model Based on Service and Process Abstractions for Management of Systems

Akhil Sahai, Vijay Machiraju
Software Technology Laboratory
HP Laboratories Palo Alto
HPL-2002-190
July 9th , 2002*

E-mail: {asahai, vijaym} @hpl.hp.com

web services,
business
processes,
modeling,
monitoring,
managed
systems,
management,
WSDL

* Internal Accession Date Only
Approved for External Publication

A Data Model Based on Service and Process Abstractions for Management of Systems

Akhil Sahai, Vijay Machiraju
HP Laboratories, 1501 Page Mill Road, Palo-Alto, CA 94034
{asahai,vijaym}@hpl.hp.com

Keywords: Web Services, Business Processes, modeling, monitoring, managed systems, management

1 Introduction

Management of systems requires measurement data to be collected from managed systems. The measurement data thus collected has to be maintained in the form of a data model that builds relationships between components of the managed system. For example, SNMP [1], a protocol for direct management of systems through agents, defined a Management Information Base (MIB). A MIB is a set of attributes for describing the properties of a managed system. The approach is simple, as the name suggests: represent systems uniformly irrespective of what is actually being managed. The simplicity of SNMP is also its limitation. Often to manage systems, it is important to capture more complex information about the system such as information about operations, sequencing of interactions, and composition relationships between components. Common Information Model (CIM) [2], on the other hand, presents an object-oriented data model that captures complex relationships between managed components. However, the complexity of CIM is its drawback. Every managed system (e.g., network, machine, and application) is modeled differently in CIM and considering all the complex relationships that have to be established between these components, the model becomes very complex. We believe that a data model based on *service* and *process* abstractions enables us to model any managed system generically. Such a model lets us develop a management system that is more powerful than one that is based on SNMP and simpler than one that is based on CIM.

The notion of a *service* as a set of operations is not particularly new. But using this as a core abstraction for automation and management of *all* systems is. Treating all systems as services that deliver some utility to other services is the fundamental principle behind the service abstraction. It lets us clearly model the *outward-facing aspects* of a system and is hence useful in interconnecting heterogeneous systems as well as in managing systems to quality of experience [3]. Web services are an embodiment of this abstraction for interconnecting applications on the Internet [4]. Web services definition language (WSDL) [5] is a standard for defining web services.

The service abstraction by itself is not enough to model the composition relationships and the flow of interactions between services. Modeling these is essential for representing the *internal*

aspects of a service, which in turn is essential for automating service execution as well as for realizing automatic analysis and drill-down capabilities in a management system. Business processes [6] are a well-understood technology for representing the flow of work in an enterprise. We use the notion of a *process* as a flow of activities in order to represent the internal aspects of a service. Web services flow language (WSFL) [7] is a business-process-like standard for representing the flow of interactions between web services.

We have developed a data model that uses these two abstractions – services and processes - while building a *web services management system*. In the course we discovered that the same data model could be used for modeling any complex system. In this paper, we present our generic management data model. The data model we propose is extensible in the sense that it provides a basic set of attributes and enables creation of new attributes. Powerful management systems that are capable of computing metrics (mathematical functions on data collected), measuring SLAs (metrics within bounds), and enforcing policies (actions coupled to SLAs) can be built on top of this data model. The rest of this paper is organized as follows: Section 2 explains service and process abstractions in detail and presents our management data model. Section 3 articulates the benefits of this model. In section 4, we present a few examples to show how managed systems can be managed through this data model. The wide variety of systems being managed demonstrates the generic nature of our model. Section 5 presents a management system that we have built for computing metrics on the top of our data model. We summarize our conclusions and point at directions for future research in section 6.

2 Modeling systems as services and processes

2.1 Services

Web services definition standards such as WSDL do a good job at identifying all the abstractions of a service. We borrow the terminology from WSDL in presenting our service data model. A *service* is a collection of *end-points*. The meaning of an end-point is similar to that of an interface in object-oriented languages. It is a collection of *operations* (methods or functions in object-oriented languages). An operation can be represented as one or two *messages* that are exchanged between the service and the user of the service[1].

Depending on the number of messages and their order, an operation can be *request-response* operation (one message from user followed by one message from service), *solicit-response* (one message from service followed by one message from user), *notification* (one message from service to user), and one-way (one message from user to service). Request-response and solicit-response operations can further be classified as synchronous (second message is "returned" in response to the first) or asynchronous (second message is "sent" in response to the first).

[1] The user of a service is another service.

For example, Unix operating system (Figure 1) on a machine can be viewed as a service with operations like getTaskPriority and setTaskPriority. Both these operations belong to an end-point called Tasks. Other operations would be modeled in CPU, Memory, and Users end-points. Similarly, a router can be modeled as a service that exposes a set of operations like routePacket and getStatus. More examples of services and end-points can be found in section 4.

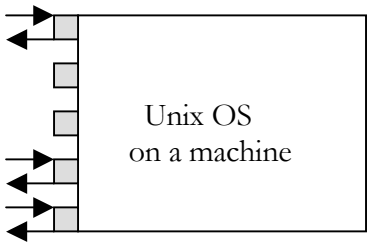

Figure 1. Unix operating system modeled as a service

2.2 Processes

In order to build management systems that can understand cause-effect relationships of problems in components (e.g., root-cause analysis), it is important to model the dependencies between services. One simple way to do this is to represent composition relationships (or "uses" relationships) between services. For example, a Travel web service uses a Payment web service. This kind of model still does not capture the order or sequencing of interactions between the composite service and the component services, which is required (see section 5) for developing management systems that can do automatic analysis and inference.

The *process* abstraction captures such relationships between composite services. A process is a flow of *activities*, where each activity is handled by executing an operation on a service. The binding between an activity and a service operation can be established at process design time (static composition) or at run-time through a policy or a mediator (dynamic composition). The process itself is initiated by invoking an operation of a service. Thus, a process both implements an operation as well as uses other operations.

This notion of a process is commonly used in enterprise workflow systems, where each activity of a process is either handled by humans or is executed by a piece of software. We make the notion of an activity more precise by connecting it to service operations. Viewed this way, a process expresses how services (more specifically, operations) are composed of each other and use each other. Sometimes, this process is publicized by a service provider to all the participating services; in other cases it is not. In the former case, depending on how much of the process is publicized, the term global flow [4] (only connectors between operations are exposed, but not sequencing), global process (entire process is publicized), or conversation

model [11] (states of interaction between services are exposed) are used to describe the public process or its variants.

Figure 2 shows a sample process that is executed when an operation called "getQuote" is invoked on a service. Notice how the process determines the order of activities and the service/operation that has to be invoked for fulfilling an activity.

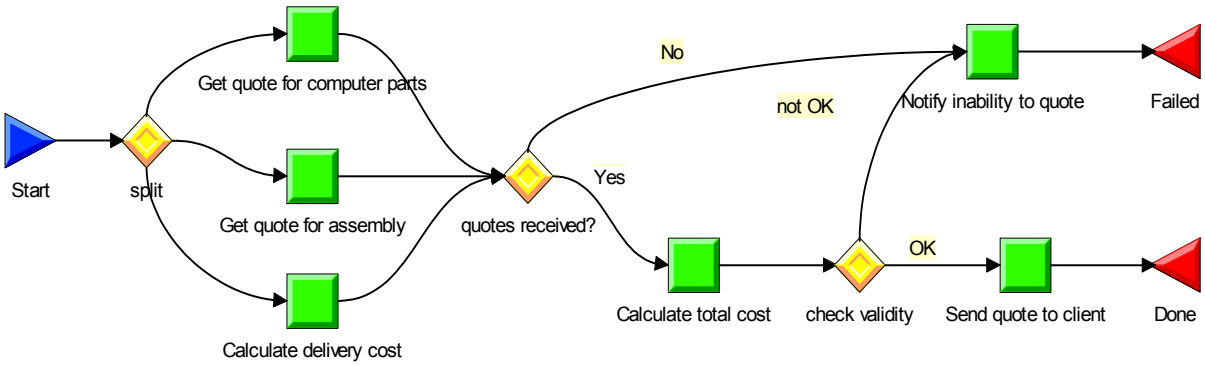

Figure 2. A process behind a getQuote operation.

While it is natural to capture process information at higher levels in the software stack (e.g., business processes and workflows at business and application level), it is quite uncommon to do the same at lower levels in the software stack (e.g., the process invoked when a getTaskPriority operation is called on an operating system). However, the process model is implicitly coded in the implementation of such operations. We believe that the lack of an explicit process model is the key reason for why it has been hard to bridge the gap between management systems at various levels in the software stack. Management systems are still not capable of automatically relating business and application related measures and events to those at system and machine level. Having a commonly used process model throughout would help solve this problem.

2.3 Management data model

Having described the two abstractions – services and process – we are now ready to present our management data model. We show the data model as a UML diagram. Classes from the UML diagram can readily be transformed into objects in an object-oriented repository or into tables in a relational database. A particular implementation of this model that we used in building a management system is described in section 4.

We distinguish between two sets of classes in the data model. One set is used to describe the "types" of managed systems whereas the other is used to describe "instances" of managed systems. Figure 3 shows the data model for the managed system types. Each of the classes in

this data model is suffixed by the word "Type". By instantiating classes in this figure, one would create new types of managed systems. For example,

Organizations or individuals create systems for achieving certain business objectives[2]. These systems provide functionalities so as to be useful to the businesses. These useful systems could be viewed as exposing a set of operations that they support. These operations in turn are invoked by consumers of these systems. These systems also sometimes have internal processes that are inter-linked with one or more exposed operations. These systems help in achieving business end-goals. Managing such systems means managing the business, processes and operations such systems expose.

The web service abstractions (namely, that of service provider, service, end point, operations, messages as captured in WSDL) and business process abstractions (processFlow, activities, links, globalFlow) are fairly general and can be mapped to any useful system. A management model can be defined on the basic web service and business process abstractions that will enable management of systems in a generic manner. This approach is similar to the SNMP approach where a MIB is defined for managed systems. These MIBs define a set of attributes on which get and set operations can be performed. However, these MIBs are often handicapped by the amount of information they can possibly expose and because of the

Business	An organization that executes business processes. The business marks the boundaries of an administrator's *domain of responsibility*. A business can put out one or more service providers. A service provider controls its Business Process Flows.
Service provider	A service provider provides services and has its functionalities implemented as Business Process Flows.
Service	A service comprises of multiple end points
EndPoint	A collection of operations combined with a binding, protocol and address for access
ProcessFlow	A sequence of one or more workflow *activities* that achieve some intended purpose on behalf of the business.
Activity	Logical entities that form a workflow. Is realized by one or more applications and exposed as one or more operations
Link	A link connects activities. These links could be data links or control links.
Operation	Exposed part of the activities in a WSDL description. An activity could also be implemented as an operation.
Message	An Operation is made up of one or more messages
Part	Part of the message. A message can have multiple parts.
User	A specific business, which invokes operations. A user could be a service provider too in a B2B scenario.
globalFlow	A set of service providers can link up their operations through plug links. The plug link links two operations from different service providers
plugLink	A plug link connects two different operations from two service providers. This helps implement conversation, collaborative process models between web services.

[2] These objectives could be driven by profit or non-profit requirements

limited control functionality they offer.

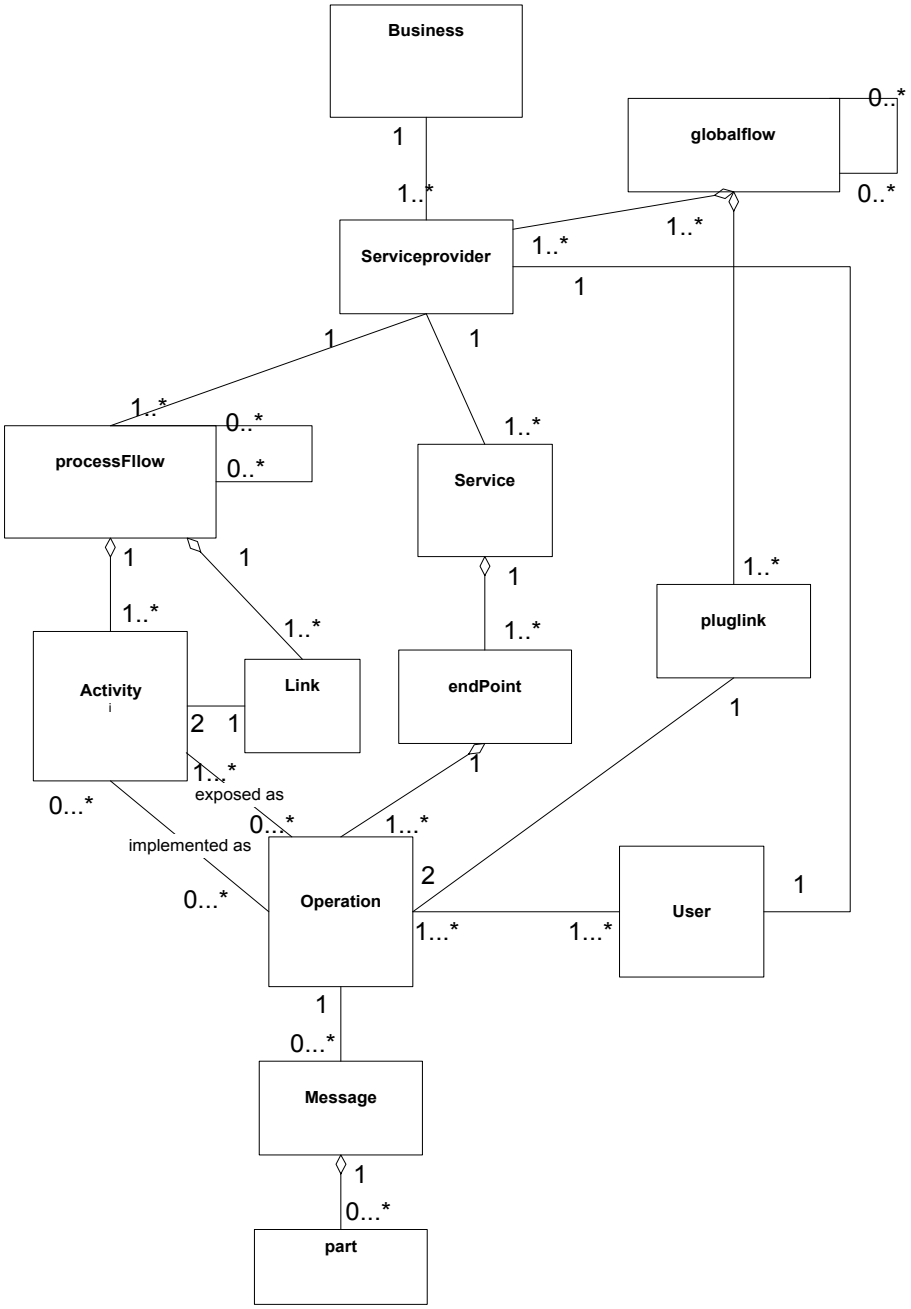

Fig 3: The basic WS+BP model

3 Managed Object Model

The intent of the basic managed object model is to create a simplified measurement model for managing systems based on web service and business process abstractions. In our model, the basic web service and business process constructs can be viewed as derived from a base class. We term the base class as the *managed object*. Every managed object has a set of *attributes*. An attribute is defined in the *attribute definition*. The attribute definition comprises of the *identifier*, *name*, *datatype*, *calculable*, *units* of the attribute. The identifier uniquely refers to an attribute definition while the name provides a label for it. The permissible data types are namely,

- counter,
  - counter-threshold
- gauge,
  - gauge-threshold
- opaque
  - boolean
  - integer
    - uInt16
    - uInt32
  - string
  - timeTicks
- uri,
  - objectId
  - url
  - physAddress
    - ipAddress
    - netAddress
    - nsapAddress

Calculable determines whether an attribute conforming to the definition will be summable. There are three different values possible for calculable, namely non-calculable, summable and non-summable. Non-calculable attributes are those that cannot be calculated (e.g. strings). Summable attributes are those that can be summed over multiple instance values. Units is a string that defines the specific units of the attribute (Bytes, ms..). New attributes can be defined by creating new attribute definitions and attaching them to the managed objects. This enables extensibility of the managed object model.

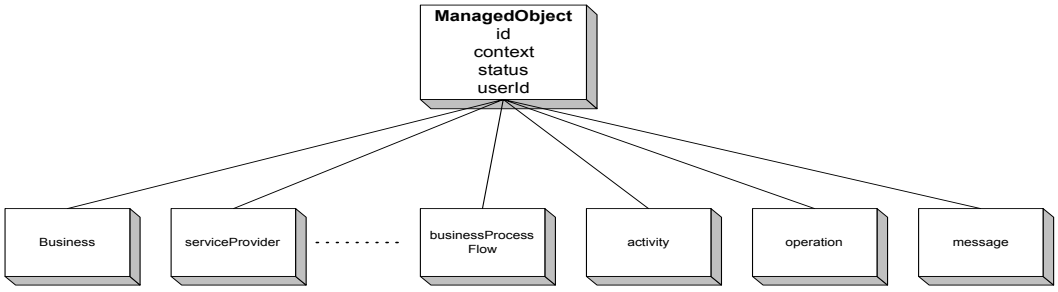

Figure 4: Hierarchy of managed object class and other web service constructs

The managed object has the *base attributes* of id, context, status, userId. All the other constructs, like operations, activity, processFlow, globalFlow, .. etc extend managed object. All the constructs thus have id, attribute, context, status, userId and other attributes that are specific to them. The additional attributes that would need to be measured at the different web service constructs (in addition to the base attributes) are shown in figure4.

4 Extensibility of the Managed Object Model

The basic managed object model is extensible. At each of the constructs new attributes conforming to the data types mentioned above can be defined through new attribute definitions. This will allow for management systems that are capable of collecting additional information about the constructs. Also derived attributes can be defined that manipulate the base attributes.

In addition, metrics can be defined on top of the managed object model as defined in the previous section. A management system may create a metric object for modeling a (set of) managed object(s). The ITU-T model is quite applicable in our case of managed systems modeled through web service and business process abstractions [6]. The ITU-T metric object model for example provides for definition of mean monitor, moving average mean monitor. Mean and variance monitor, mean and percentile, mean and min max monitor.

In order to understand the applicability of the managed object model in various scenarios lets consider certain examples.

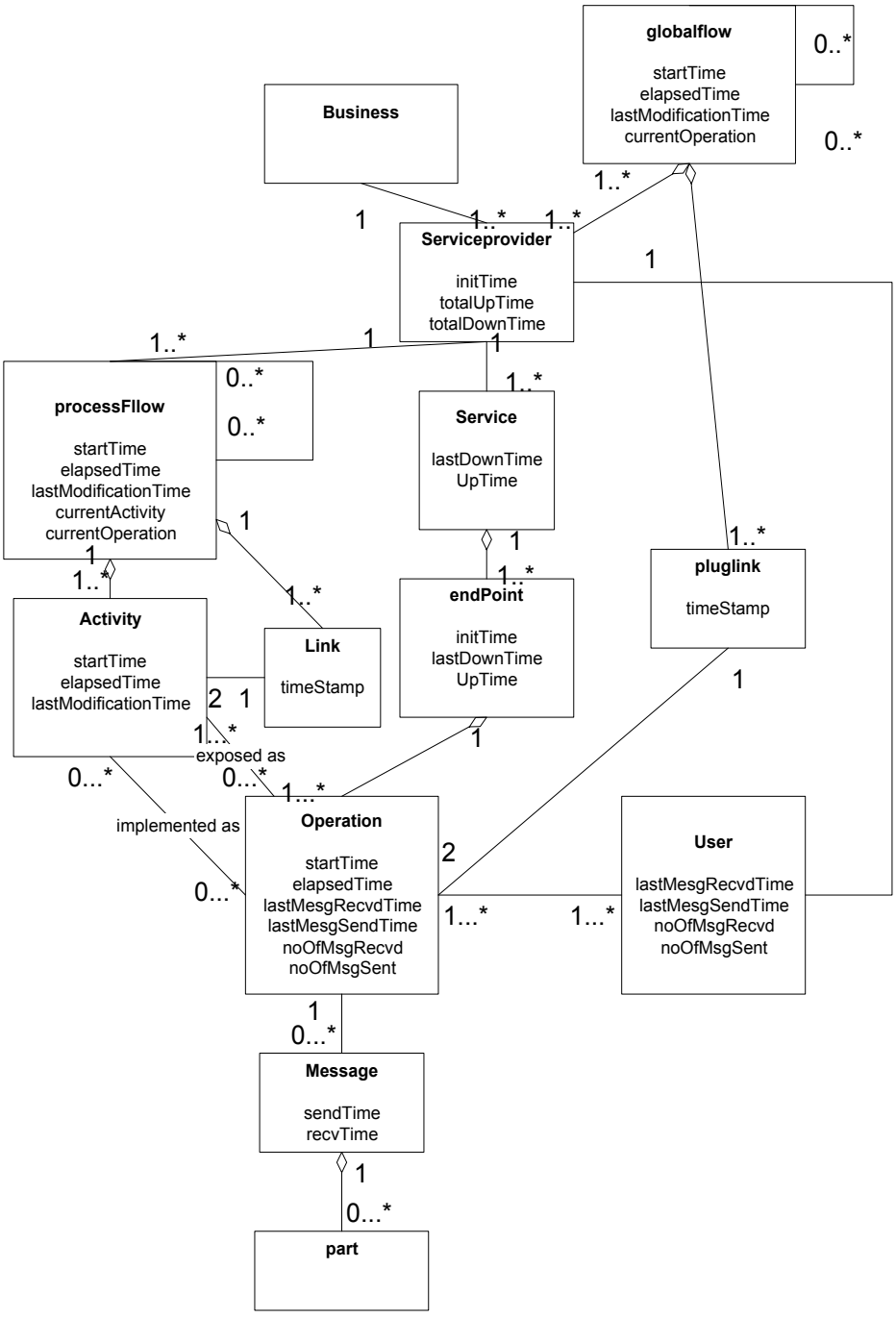

Figure 5: Managed Object model

5 Examples

5.1 Example1

The following shows a flow between a service provider (stationary.com) and the customer (officeSupplies.com). We are interested in creating a management system for Stationary.com so we will model stationary.com in terms of our general WS+BP model in Figure 7.

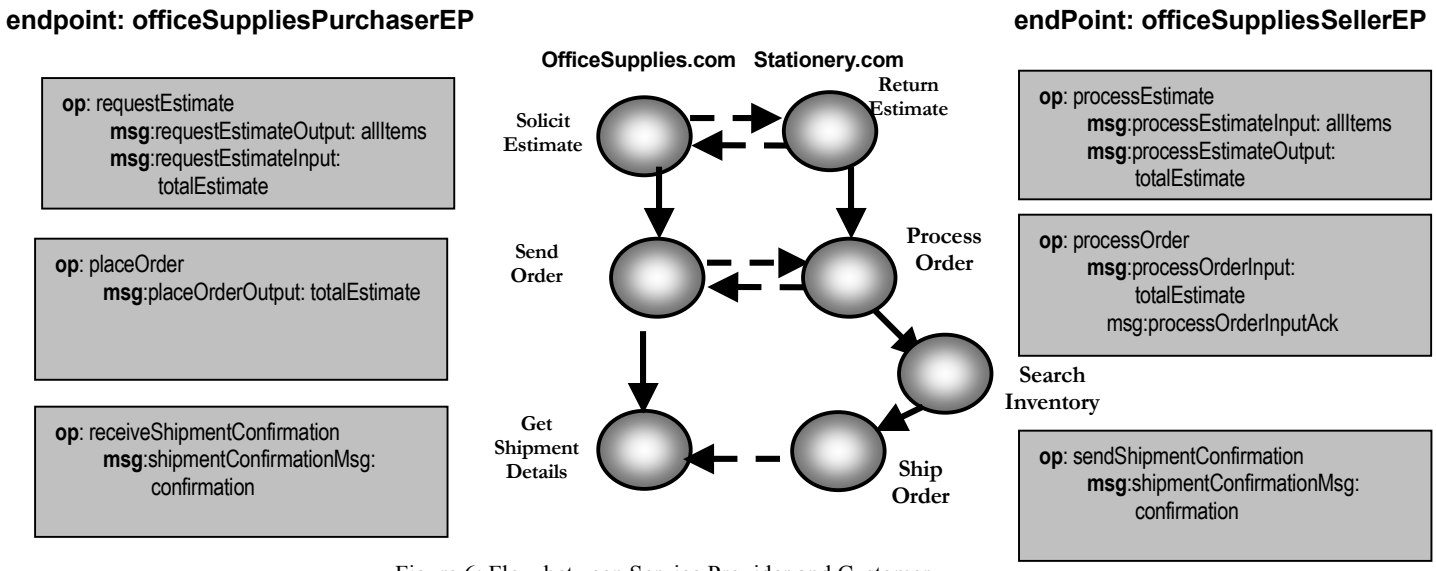

Figure 6: Flow between Service Provider and Customer

We can create a managed object model for stationary.com, which will have the basic attributes for the managed objects as shown in Figure 7.

Assuming that Stationery.com has established a Service Level Agreement with OfficeSupplies.com based on the WSDL and WSFL defined by these two services. The focus of this SLA 1 is on **four** key performance agreements:

1. ***Stationery.com will guarantee an average response time to estimate requests less than 5 minutes.***
2. ***Stationery.com will provide 100% availability to all "estimate" requests, Monday through Friday.***
3. ***Stationery.com will provide 90% availability to all "process order" requests, Monday through Friday.***
4. ***Stationery.com will process and ship orders in less than 24 hours of receiving the "process order" request.***

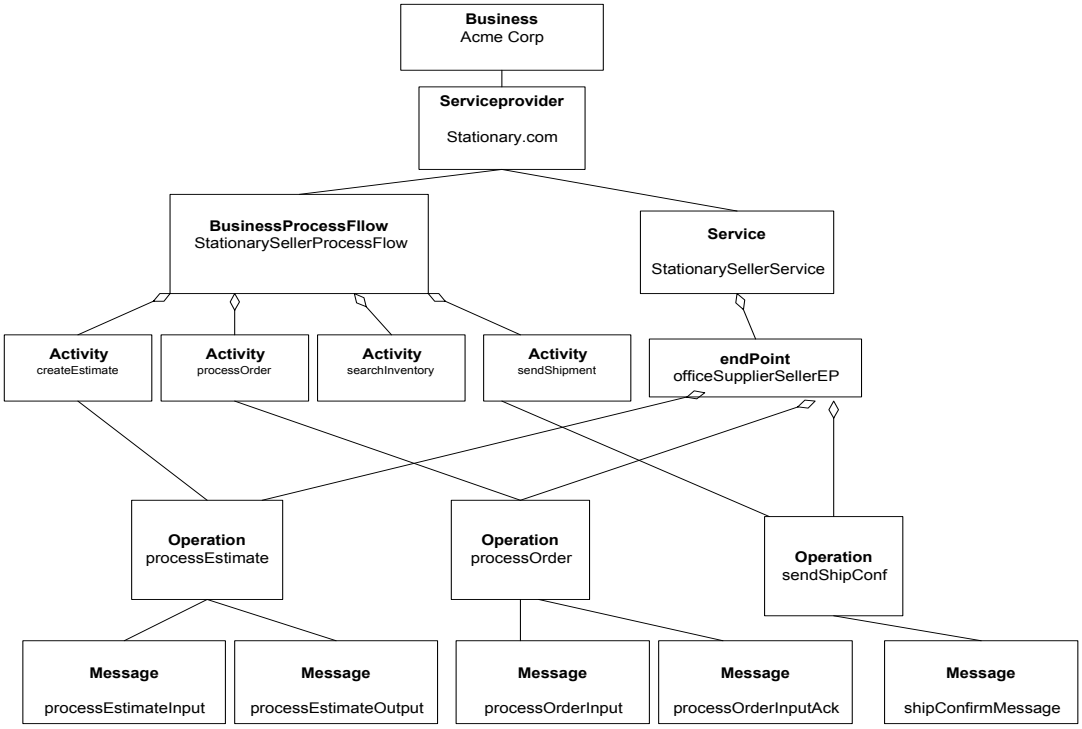

Figure 7: WS+BP Model for Stationary.com

In order to enable Stationary.com to manage its infrastructure and detect violations of any clauses, Stationary.com management system has to guarantee the following

SLO 1 : For all "estimate" requests the average of response time (the time elapsed between receiveTime of processEstimateInput and sendTime of processEstimateOutput) should be less than 5 minutes. This will mean creation of a mean metric monitor on the derived gauge attribute (difference of gauges namely receiveTime of processEstimateInput and sendTime of processEstimateOutput)

SLO 2: All the processEstimateInput must be responded to with processEstimateOutput requests Monday to Friday. This will mean scanning no of processEstimateInput messages sent and no of processOutputMessages received. These are basic attributes. They can be matched against each other to determine whether responses were sent to every request or not.

SLO 3: This example is similar to SLO2. This SLO is ensured by monitoring that for all processOrderInput messages received at least 90% of them must be replied to with processOrderInputAck messages.

SLO 4*:* This SLO is ensured by making sure that for all instances the time elapsed between receiveTime of processOrderInput message and sendTime of sendConfirmShip message should be less than 24 hours.

Let us consider some internal SLAs. Stationary.com also has an internal activity that of searching inventory. It has no corresponding operation (see Figure 7). This particular activity locates the product stocks in the inventory and then ships it. The management system for stationary.com also needs to monitor the activity "search inventory" so as to guarantee the following internal SLA

1. ***Stationery.com will search Inventory in not more than 6 hours***

For monitoring the internal SLO, however it needs to do the following

SLO 1: The time elapsed between startTime and endTime of activity search Inventory should not be ever more than 10 seconds

5.2 Example 2

The managed object model can be used to capture system and resource management processes involved. Assume the organization ACME has about 10,000 Unix machines, 15,000 NT based machines, 500 Linux Machines, connected by an enterprise network comprised of large number of routers, hubs, repeaters that form backbone of the network connecting these machines (Figure 7). There are also processes that are executed at regular intervals of time. The processes are versioning processes that update the software on each of these machines, do a regular weekly backup of data on these machines into a back-end storage area network, other administrative processes. There are also processes for outdating certain machines at regular intervals of time and replacing them with new ones either on request or as a part of policy.

The machines (NT, Unix, Linux based), the network hardware (repeaters, hubs, routers) can be all visualized as services that have well defined operations. Desktop machines can be modeled as services that have operations (namely, getCurrentUsers, getAdmin, setAdmin, getCPUUsage, getMemUsage…) with attributes like (topProcesses, users, administrator,

OSType, IPAddress, sysUpTime, , sysDescr, purchaseDate, ….) defined on top of them. Similarly network hardware can be visualized as services with attributes(like sysUpTime, ifPhysicalAddress, ifAdminStatus, ifOperStatus, ifInOctets, IfInUcastPkts, ..) and get and set operations for these attributes (getsysUptime, getifPhysAddress…).

The processes can be again modeled in terms of business processes with activities for software version update, for syncing up a storage area network with the data stored on these machines, or even a process based on a policy that involves changing machines at regular intervals of time (by basing decision on purchaseDate attributes of a machine).

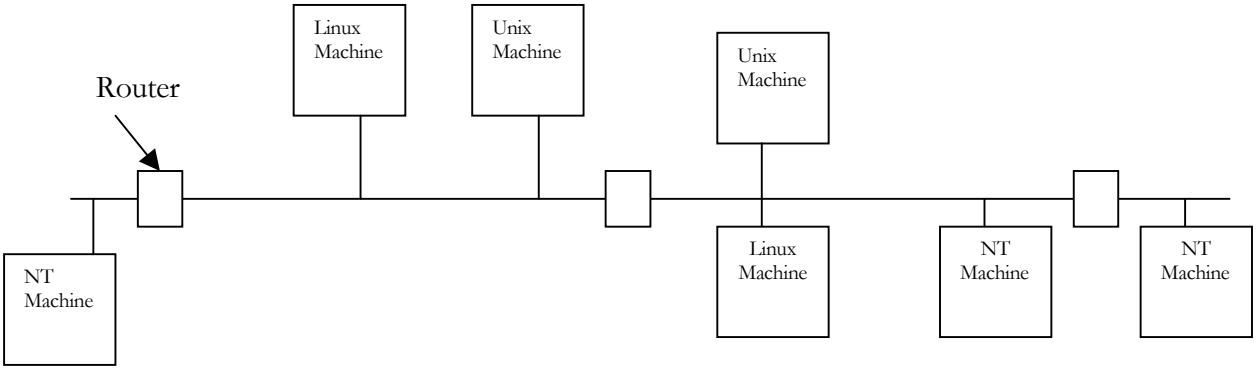

Figure 8: Enterprise network of ACME corp

5.3 Example 3

In this example we will look at the Internet infrastructure of Acme. The internet infrastructure will comprise of web server farm, a set of application servers (in this case J2EE based), and a storage area network (Figure 9). Clients connect through the web server farm to certain EJBs on one or more of the J2EE Application Server.

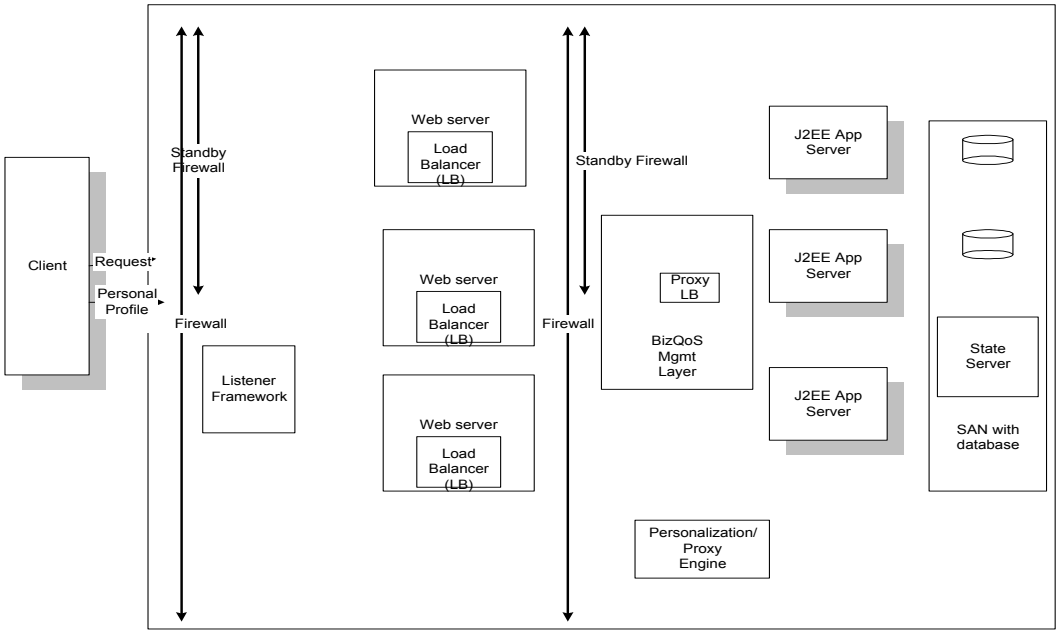

Figure 9: Internet infrastructure of ACME corp.

Our managed object model can be used to model the internet infrastructure. The web servers can be each modeled as a service with attributes (no of connections, lastDownTime, UpTime, administrator, averageSizeOfPayload, ..) and operations to get and set them. The application servers can be modeled again as services with attributes (noOfEJBsActive, noOfServicesHosted, J2EEVersion…) and operations like getAllHostedServices, getNumberOfbeansForService, getActiveBeans, getStatusOfBean….The SAN again can be modeled as a service with attributes and operations.

Certain processes are also involved in the Internet infrastructure. These involve the load balancing process for routing transactions, adding or removing additional application servers and web servers depending on the load, failover to standby databases in case of failure in the SAN. All these processes can be modeled as business processes with corresponding activities.

6 Related Work

SNMP [1], though meant for direct management of systems through SNMP agents, did that by defining standard interfaces in terms of Management Information Base (MIB) for management of varied systems, ranging from network router, repeaters to machines. The approach is simple, as the name suggests represent resources uniformly irrespective of what actually is being managed. The simplicity of SNMP also is its limitation. Often to manage systems, it is important to capture more complex information about the system like the sequence of operations that are performed. It is important thus to capture the process aspects

of the model which SNMP fails to do. Also SNMP MIB(s) are typically focused on the Network Element Layer and are not used for provisioning and configuration, nor is SNMP implemented in all problem domains (that is due to the fact that expressing complex relationships requires a complex object-oriented model rather than a simple list of name-value pairs).

CIM [4] is a fairly complex object-oriented model that is used for modeling managed systems. The object hierarchy is rooted by managed element. The CIM Core Schema abstractions are: ManagedElement, Collection, Setting, StatisticalInformation, PhysicalElement, LogicalElement, LogicalDevice, System, Service, and ServiceAccessPoint. Collection is used for grouping instance data into category bags. Statistical Information class is used for modeling statistical information on the ManagedElement. It is also desirable to have a clean separation between describing the physical world and the logical world. PhysicalElement is the parent of the class hierarchy that describes the physical world (things that adhere to the laws of physics that can be seen or touched). Logical Element is where most of the management occurs. The sub-classes of LogicalElement are LogicalDevice, System, Service, ServiceAccessPoint.
CIM takes a bottom-up approach trying to model every managed element as a class in the model. Though there are some core abstractions (as described above) in CIM code model, the real way that management systems are built are around CIM's extension models, which vary depending on the managed system (for e.g., CIM has different extension models for network, system, applications, etc). This once again makes building management systems quite complex. Another drawback of CIM is that it does not capture the abstraction of process, which is essential for performing complex management tasks.

Conclusion

The intent of this article is to present a simple model for management of systems. This model is based on the service and process abstractions. By creating a managed object model and a metric object model, these managed systems can be monitored and controlled based on the information collected. The simple model enables virtualization of managed systems and will enable creation of generic management systems based on this model.

Acknowledgements

We would like to thank Aad van Moorsel for his help and insights in developing the ideas presented here. We would also like to thank Mehmet Sayal for his inputs.

References

1. Simple Object Access Protocol (SOAP) . http://www.w3.org/TR/SOAP
2. Universal Description Discovery Integration UDDI . http://www.uddi.org
3. Web Services Description Language (WSDL) Specification http://www.w3c.org/TR/wsdl
4. Web Services Flow Language (WSFL) specification. http://www.ibm.com/webservices/
5. Simple Network Management Protocol (SNMP). http://www.snmp.com
6. Recommendation X.721 at ITU-T http://www.itu.int/
7. Application Response Measurement (ARM) http://www.opengroup.org/management/arm.htm
8. Common Information Model (CIM) specification at DMTF http://www.dmtf.org/education/whitepapers.php
9. Java Management Extensions (JMX) http://java.sun.com/products/JavaManagement/
10. XLANG at Microsoft http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm
11. Web Service Conversation Language (WSCL) http://www.w3.org/TR/wscl10/