Jiro™ Storage Management

Bruce K. Haddon, Ph.D.
The Java Centers, Sun Microsystems, Inc.
500 Eldorado Boulevard, UBRM01
Broomfield, CO 80021-3400, U.S.A.
Bruce.Haddon@sun.com
tel +1-303-272-8418
fax +1-303-272-5011

William H. Connor, Ph.D.
Network Storage, Sun Microsystems, Inc.
2990 Center Green Court South, UBOL03
Boulder, CO 80803-2216, U.S.A.
William.Connor@sun.com
tel +1-303-272-8414
fax +1-303-272-8427

Abstract

The *Jiro*™ technology provides an environment intended for the implementation of storage management solutions. A product based on Jiro technology is an implementation based on the *Federated Management Architecture (FMA) Specification*, which describes extensions to the *Java*™ language environment. The FMA initiative addresses system management, particularly storage management. In addition to the platform, the FMA Specification defines a component model, *i.e.*, the *FederatedBeans*™ model, and a set of services. The Jiro technology is effectively the application of this model to the design and implementation of storage management solutions.

The FMA assumes a three-tier architecture for the design of storage management applications: the first or top tier is the client/presentation layer, or interaction layer with user's or systems acting as a client of the storage management application; the third or lowest tier represents the storage and related resources being managed; and the second or middle tier is that containing the logic (the programs) that define and effect the management actions required by the user upon the storage resources. It is the middle tier in which the FederatedBeans components are deployed.

FederatedBeans components are each implementations of the concept of a *Jini*™ service. Each FederatedBeans component is an embodiment of some function that provides a service to other entities in the second and first tier. The success of the initiative surrounding the Jiro technology will be availability of a wide variety of FederatedBeans components from different suppliers, each providing significant functionality for the construction of storage management applications. The FMA platform supports automated communication between networked Java Virtual Machines, thus promoting applications that are federations of the constituent components.

Within FMA, the resources being managed are expected to conform to the *Common Information Model* (CIM), although provision is made for the management of resources by other means. The CIM provides modeling for all common storage system elements.

1 Introduction

Today's sciences and businesses depend on information—a vast mountain of information. As the demand for storage to hold all of this continues to grow at phenomenal rates, the management challenges are growing too [1]. In fact, the diversity of installed storage systems and the wide distribution of those systems are on the verge of creating a crisis in management, sometimes described as a "nightmare."

The amount of management that needs to be done is also increasing [2]. The cost of management is one of the significant areas of rising cost in using computer systems. Since management is a continuing cost, it has a large influence on the assessment of "return on investment" (ROI). All aspects of a system need management: software, processors, boards, options, network connections, storage devices and networks, modems, printers, and all the other pieces that are put together to make a "system." Management, in this context, includes installation, configuration, asset deployment and inventory, performance monitoring, error and failure detection, upgrade and replacement; as well as more difficult things like capacity planning, service level contracts, load balancing, all of which may be controlled by policy decisions made by the owning organization.

Because storage and storage subsystems are amongst the most complex parts of the management problem, as the storage systems often including processors, switches, and storage area networks, effective solutions to storage management problems are urgently required. Thus, the initiative surrounding the Jiro technology focuses its attention on storage management.

Developers lack standard middleware infrastructures for efficiently building capable solutions for heterogeneous management tools, applications, and services. Further complicating this situation, to provide storage solutions, developers must port their products to multiple proprietary platforms, a costly, time consuming process. The answer is based on building with software components designed for a platform specific to the purpose, being an open management platform based on Java [3] and Jini [4] technologies.

1.1 The Jiro Technology Solution

The need for the Jiro technology is acute due to a storage landscape dominated by point products that do not interoperate, creating large islands of information that are difficult to integrate, complex to manage, or that actually prohibit cross-platform information management. The intent of the Jiro technology is to bring the benefits of community source processes to the development of storage management solutions. This platform, together with basic services and a component model, brings an order to the creation of the software that can automate or add intelligence to all management functions. The elements and their relationship are defined in the Federated Management Architecture (FMA) [5].

The history of the Jiro technology activities starts with a proposal made to develop a Java language extension designed to make it easier for the developer to create new storage management applications, enable faster design cycles, lower development costs, and offer a wider market potential. It is further intended to alleviate the need to conform to multiple API's and interface specifications. Following the requirements of the *Java Community Process*$^{\textbf{SM}}$ (JCP) [6], a call for experts (CAFE) was issued, followed by the formation of the Expert Group, a work group made up of representatives from a number of interested industry leaders. The Group was convened under the terms of the Process, and the FMA Specification is the result of the work of that Group.

1.2 The High Level Architecture

Three areas of specialization can be seen in the management problem:

- the representation of the resources to be managed, including the management data they contain, any behavior they exhibit, and a means of understanding their topology and other interrelationships;
- the interaction with the wishes of the user or users of these resources, including all means of invoking and scheduling management activities; and,
- the computational logic that is needed in order to translate the wishes of the users into the desired actions on, or using, the resources, or, as importantly, the translation of events taking place within the collection of resources into the presentation of useful analyses to a representative of the users.

In the diagram below (Figure 1), the third tier in reality comprises software objects, some representing the actual hardware and software being managed (solid rectangles), and some representing the relationships between them (striped rectangles). Every entity in the domain being managed, whether hardware (*e.g.*, a disk), or software (*e.g.*, a database manager), is modeled by a representative object. This view of management sited beside the elements of the data stack has been proposed earlier [7].

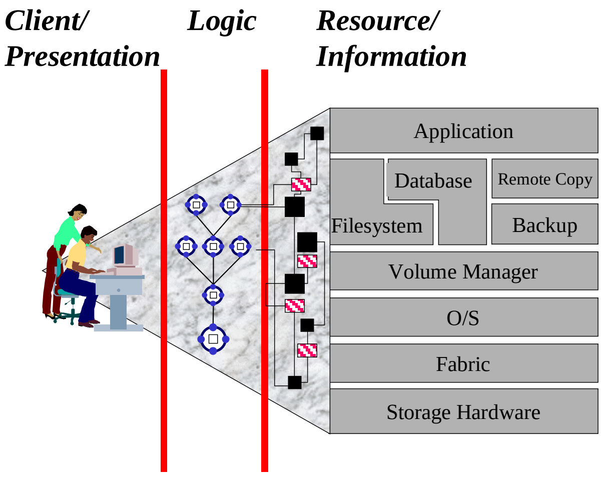

Figure 1

These are the requirements that, in other enterprise, business, and Internet applications, have led to the adoption of what are now known as “three tier” architectures. A pictorial depiction of the three-tier architecture as it applies to management is shown above.

1.3 The resource/information tier

In storage management, the resource/information tier contains both the physical hardware used to create the storage resources: the disks, tapes, disk subsystems, automated libraryies, channel interconnects, storage area networks, and so on, as well as the logical elements: the storage extents upon the media, logical disks, volumes and virtual volumes, file systems, databases, and so on. The ***management*** information contained in this tier includes the attributes of each of the pieces of storage, such as its size, capacity (which may be larger or smaller than size, depending upon reserved areas of the media, and compression, *etc.*), speed, access time, geometry, and a large number of other attributes.

Not so obviously, the management information also includes the relationships between the resources. These relationships include, for example, how the resources are connected to each other or the host, to hubs and switches involved in creating network segments, and which media are involved in the realization of a file system. The relationships are not only important to the management software for monitoring the behavior of those resources, but often are the elements most directly being managed. For example, if a particular interface card fails, it is be desirable that the monitoring management software be able to reroute traffic *via* another interface, should there be another interface available.

This third tier must be considered "active." Therefore, the representation of the resources must be able to define behaviors of the resources in a variety of ways. An example is the formatting of a disk—a common behavior, but perhaps implemented differently on different manufacturer's disks. The definition of these behaviors, taken together with the attributes of the resources and the information implied by the relationships between them, is the management interface of the resources, and hence of the resource/information tier.

In the three-tier model, the third tier represents the "state" of the system, in this case, the resources of the management system. In the case of storage resources, the state is the sum of the management data and relationship information.

There is a very critical relationship between the resources being managed and the application software. Undoubtedly, the whole object of storage management is to enable and optimize the delivery of data to those applications. Application data usually go through a number of paths and transformations in moving from "raw" bits on the storage media, to the form in which they are presented to the application and even further transformations in order to present information to users. Therefore, "behind" the resource representations as seen by the management logic, there are layers of hardware and software responsible for these data deliveries and transformations. Figure 1 also shows this.

1.4 The client/console

In the first diagram, the client/console tier has been represented by a system administrator or installation manger sitting, literally, at a console. This depiction is only meant to be representative. Certainly one of the objectives of enterprise-class management systems is to bring relevant monitoring information to some central place, where the "big picture" may be evaluated effectively and efficiently.

For an automated management system, it is not sufficient that operations be initiated from a console. It is necessary to be able to initiate actions from CLI's, scripts, periodic schedulers, and components of the management system itself. This latter facility is required so that when decisions are programmed into the management system, those decisions may be used to initiate appropriate actions, without the real-time intervention of an administrator (which is the definition of "automated management").

1.5 The management logic tier

This tier is intended to contain the actual programs that are the applications implementing the logic needed to perform management decisions and management functions.

Such applications typically contain a number of modules, architectural "sub-applications," that deal with various aspects of the overall tasks to be performed, by doing an individual task, or computing a specific result, and so on. Component models, based on object-oriented methodologies, and usually based on specific object implementations, are a way of describing these modules. The FMA specifically defines such a component model, *i.e.*, FederatedBeans.

All component models, in essence, define a "platform" and a set of "services" in addition to the component model itself. A number of other things also normally surround the successful use of the component model for real implementations, which includes some or all of an interface specification language, an intercommunication protocol, a deployment methodology, and various tools to support the creation and use of the component model, which together constitute a software development kit.

However, more than each of those, a successful component model is one that leads and encourages implementations of actual, real, useful components, that is, a library of components, which implementers may use without further development.

The largest development leverage lies in being able to obtain significant components out of which an application may be built for appropriate amounts of money, while capping the time and effort required to learn how to use and to support them (when deployed). The other effect is that the components may be common to more than one application. Thus a component has only to be deployed once, the footprint costs are paid once, and many applications can share the use of the component. Such components take on the nature of additional services, but without requiring a new definition of the *core*.

So, by analogy, once a platform based on Jiro technology is available, then the objective of on-going activities is building useful applications using components, and, as experience is gained, determining which of those components are candidates for libraries of re-usable components. Such candidates will be evaluated, and perhaps re-engineered, for standardization, interchangeability, substitutability, *etc.* Attention and growth in this area will eventually lead to a classification of components, and a catalog from which developers will be easily able to choose the components most suited to their current needs.

1.6 Generation of management applications

The usefulness of the three-tier architecture is related to the decoupling that occurs between the tiers. For the three-tier approach to be valid, the degree of decoupling between the tiers should, in general, be greater than the decoupling introduced by the problem or task decomposition used within the tiers.

The most usual method to present one tier to another is through API's, and most particularly API's that are wrappers for protocols. From the point of view of a lower tier presenting itself to an upper tier in the form of a particular API, the requirements of that API define a framework. That is, it requires that certain conventions be observed so that functions may be called, results passed back, call-backs created if needed, and so on.

Provided that these API’s or protocols exist, the internal structure of each tier may be driven by the needs and requirements of that tier. Consequently, the overall philosophy of the Jiro technology has been to use the “best of breed” examples for each tier. The FMA proposes the technology for the second tier, and encourages support of the DMTF CIM for the third tier. Work is continuing on defining interfaces to the first tier.

1.7 Requirements of the Third Tier

There are many requirements, in detail, for this tier. The following may be considered just the major requirements:

- a software-accessible representation of the each of the actual physical resources that are part of a computing system, including all the usual asset information (*e.g.*, manufacturer, type, size, *etc.*);
- a software-accessible representation of the system resources made available by the existence of these physical resources (*e.g.*, the storage extent actually made available by the installation of a particular storage device);
- a representation of the relationships between resources of both types (*e.g.*, this disk is part of this RAID subsystem, this file system is implemented on this stripe of these disks), and the ability to update these as the system evolves, or fails;
- an ability to recognize that various resources are implementations of the same class of resource, as well as being able to identify and take advantage of specific differences (*e.g.*, a laser printer is a (generic) printer, but is capable of duplex operation);
- a reasonable ability to find out what resources are available (*e.g.*, this particular host has a tape drive, whereas some other does not); and,
- agreement on the names and meaning of various critical attributes and behaviors, with support for being able to find the resources represented by means of queries based on those attributes.

1.8 The CIM Specification

The *Common Information Model (CIM) Specification* [8] is a description of an object model, and of a language in which to describe the classes and the instances of objects of that model. This particular object model has, as do many other object models, rules of inheritance and the overriding of methods and properties. In particular, the inheritance is single inheritance, and overriding is explicit.

Nominally, there is only one kind of object in the CIM, but it is more instructive to think in terms of two principal kinds of object. There are those objects that represent the actual entities being managed, which are those needed to satisfy the requirements above; and, those objects that represent the relationships between those entities (providing a way to satisfy other requirements, above). These objects are called *associations*. For example, a network hub and a cable of the network are each entities. The cable will have a relationship to the hub, which can be described as “connects to.” The “connects to” object contains a reference to the cable as being the source (“antecedent”) of the relationship, containing also a reference to the hub as being the target (the “dependent”) of the relationship. Of course, associations may be one-to-one, one-to-many, many-to-one, or many-to-many.

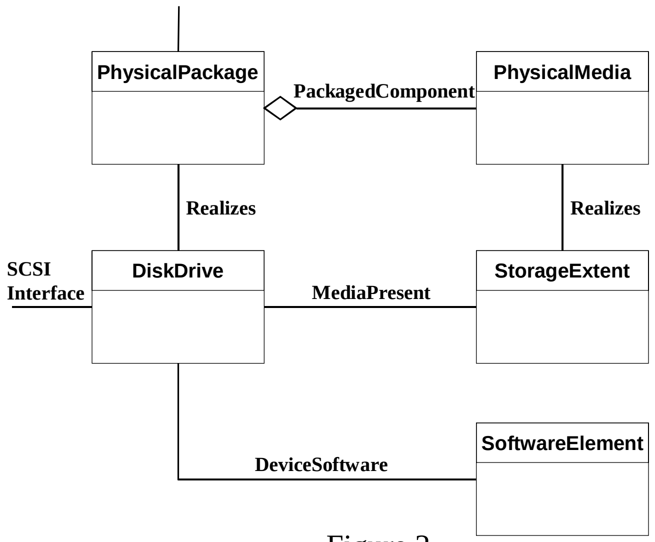

Figure 2

The diagram above (Figure 2) shows a typical model of a hard disk drive, in its own packaging. It shows the both physical and logical elements, and the relationships between them, these being the associations illustrated as labeled lines. It should be understood that each of the associations in this diagram is itself a CIM object, and thus may be queried and interrogated in the same manner as the objects representing the system elements.

1.9 The common XML protocol (WBEM)

While there is considerable value to the concept of implementation independence when defining a common information model, a plethora of implementations is not something with which developers of management implementations wish to cope. There is, however, a middle ground permitting *Object Manager* (OM) implementers freedom, but also allowing use by applications without having to re-implement their code for interacting with the OM. This middle ground is based on having a common protocol to be used to communicate between the application and the Object Managers.

The DMTF has standardized a protocol for this purpose. The content of the protocol is based on XML [9], and the use of HTTP as a transport mechanism is also defined [10]. While it is possible to use the XML "documents" directly to invoke actions upon the OM, the use of the HTTP binding allows the XML payloads to be transmitted across the Internet, and, where permitted, through firewalls. The XML format provides also for conveying results of method invocations on the Object Manager back to the OM client.

The combination of a CIM Object Manager implementation with the HTTP-transported XML protocol packet for OM operations, is "Web-Based Enterprise Management" (WBEM) [11], which is the platform- and language-independent technology for using CIM and CIM Object Managers. The diagram below (Figure 3) shows the WBEM model for the use of a CIM OM, where the communication between a client and the OM is by means of the defined XML-based protocol, and the CIM OM obtains information about the managed objects by means of privately defined code elements called "providers."

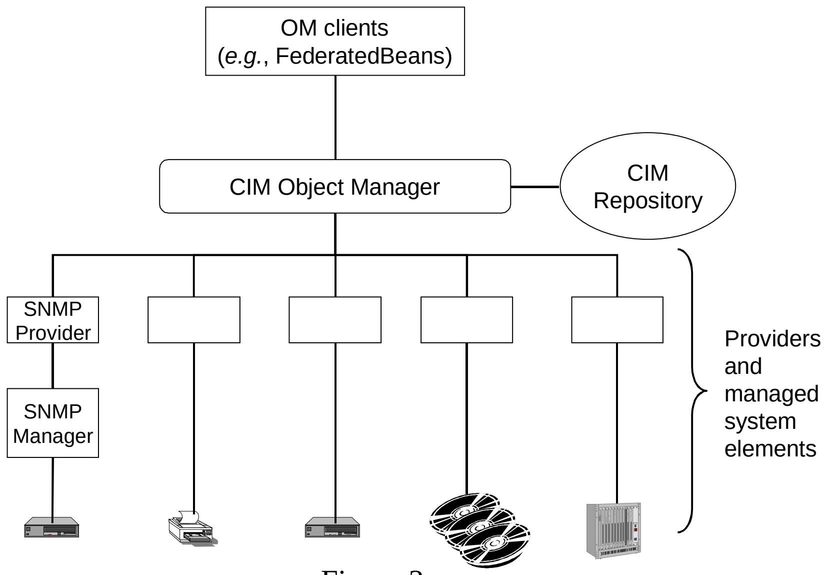

Figure 3

The existence of a CIM Object Manager implemented using the Java language is an important development. It means there can be a CIM OM available everywhere that the Jiro technology is available, with no further investment in the development of the OM. This is undoubtedly a significant advantage when developing management applications.

1.10 The requirements for the Client/Console Tier

As mentioned above, this tier is not specifically a target of the current Jiro technology activities. However, for some completeness, the requirements of this tier are outlined here, with some description of the considerations. The requirements include at least:

- the ability to interact with users, either in terms of graphical user interfaces, CLI's, and perhaps more than one means of scripting the interaction;
- support for remote communications when the user must be able to access the management system from other than a fixed location. Most newer management systems contemplate the use of the Internet as a means of access (see next item);
- provision for gathering information to enable authentication of the user, in order to be able to use the security features provided by the second and third tier;
- the possibility of supporting interfaces to existing management solutions, since there is already a great investment in these. The first tier could provide a good proportion of the bridging between such legacies and newer Jiro technology-based solutions;
- effective means of navigation for each of the user interfaces supported. Most computing systems, particularly those supporting storage area networks (SAN's) and similar large and complex subsystems, require means of showing them that can be mapped to and from the model representations that make sense to the user;
- presentation technologies, either screen or print based, to show the navigation and other views, upon demand; and,
- the capability of launching management applications into the appropriate environments, with, preferably, the ability to monitor the status of those applications.

There exist many "console" and client-side solutions, offered as complete products by many companies. In addition to the actual first tier functionality, these same companies often offer modules that provide second and third level functionality. These product offerings are sometimes embedded in other services, including monitoring, notification and response management, and sometimes even "first response" field engineering.

All of the above services remain relevant in a Jiro technology-based system. These systems offer the promise of even more capable applications, and hence even more desirable value-added services.

2 The Management Logic Tier

This is the tier specifically addressed by the FMA Specification. The Specification defines a component model for the development of management applications (programs that implement storage management). Because the Specification contains all the needed detail about this tier, only a brief overview is offered here.

2.1 Requirements for this tier

The question is often asked, given that the third tier contains so much capability, why is there a need for the second or "logic" tier. The requirements for the second tier include at least the following, each going beyond what is (conveniently) possible in the third tier:

- a component model (more detail below), in order to be able to create a library of solutions to be used as construction elements for new management applications;
- support for distributed applications, in order to support scalability (*e.g.*, use of more than one processor), efficiency (placing logic near to its source of information), enterprise capability (separation of management environments along organizational lines), and redundancy (to support applications utilizing high availability capabilities);
- deployment of components in standard ways, so that packaging is done once, the deployment problem needs not be re-solved for each release of each application and for each implementation of the second tier;
- basic services, that are needed by all applications, inclusive of component location and loading, logging, scheduling, *etc.*;
- control arbitration, whereby a component can claim sole access to a second or third tier resource, and then act as a "gatekeeper" for allowing appropriate access (*e.g.*, a classic multiple reader, single writer regime);
- an ability to ensure consistency across resources, even when those resources are in different environments (*i.e.*, namespaces), such as a remote disk mirror;
- the ability to compute logic across resources, in a similar way, such as switching between remote disk mirrors;
- a facility to share logic implementations between clients, by allowing multiple clients to re-use (or multi-thread, if appropriate) the same logic components;
- the ability to compute logic across time, by accumulating historical information about present and past states of the resource/information tier, and performing appropriate (*e.g.*, statistical) analyses; and,
- the definition of higher level of management abstractions, with appropriate interfaces (*e.g.*, a charge-by-usage service of a virtual disk system across the Internet).

2.2 Jiro Technology as the second tier

The Federated Management Architecture has been specifically proposed as a means of structuring the second tier. It addresses specifically the following characteristics:

- the ability to dynamically and easily introduce new behavior while the system is in operation;
- the necessary locking to support control arbitration (as described above);
- support for wide-spread consistency across management applications;
- transactions across arbitrary parts of the management state;
- the possibility of a single packaging, that support "talks to" relationships between components written by different authors, and a universal "runs on" relationship to the Jiro technology (thus fostering neutrality with respect to physical platforms);
- fine-grained security, in that the security context is carried and made available to all components in the distributed system;
- source available under community agreements;
- provision for the support for higher availability solutions; together with,
- support for adequate scalability of management solutions; and,
- versioning, to provide a methodology for not having to update an entire universe of solutions at one instant.

The resource/information layer (3rd tier) models that which is being managed (systems, storage, networks, *etc.*). The model includes all the manageable attributes and behavior of the resource. These attributes and behaviors of the model are "static" in the sense that they are in one-to-one correspondence with the attributes and behaviors of the real-world resource being managed. These attributes and behaviors of the model should not be changed (even if the underlying implementation technology would permit it) to represent anything other than the attributes and behaviors of the real-world resource. It is not expected that the "external logic" of a managed resource should change in any significant way during its lifetime. The resource/information layer is *not* static in the sense of being unchanging—resources are expected to come and go, being replaced, upgraded, and extended, in the normal course of the system lifecycle.

The logic layer (2nd tier) reflects the pattern or structure of the decisions that need to be made in order to manage the resources. These decisions will be made based on information that is corralled and collated from the data present in the information model. Since management is undertaken in order to meet (organizational) goals, the nature of the decisions, and the behavior based on those decisions, needs to change as the goals change or evolve. Thus, the objects (or components) of the logic layer need to be replaceable with new versions, which, behind the same API, implement new behaviors. The logic layer also needs to be "dynamic" in that an object or component may be introduced into an execution environment where none has existed before, thus defining new behavior.

2.3 The FMA Specification

The fundamental notion supported by the FMA is that of components, where it is intended that these be the unit of assembly and installation of management logic. The component model consists of a set of naming and construction rules; with these components being termed "FederatedBeans." FederatedBeans components are based on the Java object model, and conform to the set/get conventions of *JavaBeans*™ [12]. To support assembly, it must be possible to discover ways in which management

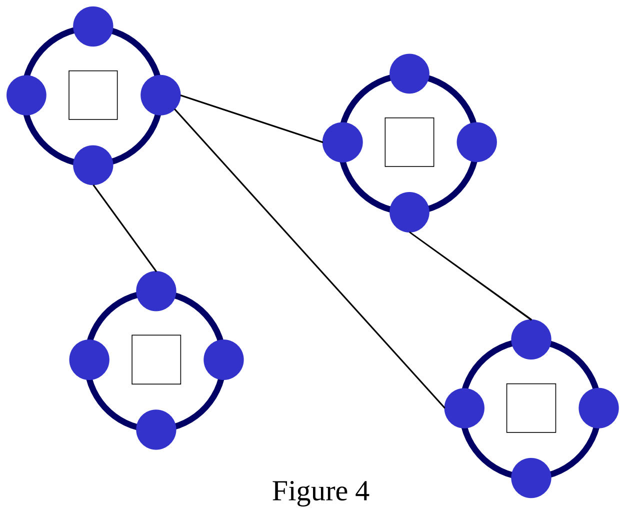

Figure 4

components can be connected to one another in both anticipated and unanticipated ways. The components have to find the appropriate interface offered by other components in order to create a coherent application. As illustrated in the diagram above (Figure 4), the connections between the FederatedBeans components may be an arbitrary topology (not necessarily hierarchical), and a given service might be used both directly and indirectly. In many programming environments, the choice of interface is made at program writing time. An effective component model will allow these to be found at installation and/or execution time. A component may present different interfaces for different purposes, or even just to provide the same functionality in more convenient forms.

Because the FMA platform supports a distributed programming environment, most frequently it is not the actual interface that is the point of connection between components. The point of connection is a *proxy* [13] for that interface. The proxy is always local to the using component (*i.e.*, present in the same *Java Virtual Machine* (JVM™) [14]), and the component (or object) for which the proxy is acting may be in the same virtual machine, or other virtual machine accessible within the domain of the application.

Deployment: It must be possible to deploy, or install, components in a standard manner on a running system. Deployment includes installing class files, resources, components, and objects.

Controllers: An important objective of the Jiro technology is providing the infrastructure to support control arbitration. Controllers attempt to control resources through components. Resources, therefore, may be subject Jiro technology providing the access mechanism to support control arbitration. The primitive required for arbitration is called the controller *aspect* of the management services model, and this in turn must support durable (long term) exclusive locking of resources.

Transactions: Most distributed component models provide some form of transaction support to aid in protecting the integrity of the resource/information layer [15]. The transactions provided by the Jiro technology are focused on supporting large numbers of heterogeneous resources, rather than a single large resource (*e.g.*, a database), and not necessarily large numbers of clients. A FederatedBeans component needs a transaction aspect to participate in a transaction.

Security: A management environment must support validating clients for actions that they attempt to take, since such actions may have far-reaching results. The basic model is that of the Java model [16], but with provision made for ensuring that necessary security information is transmitted between Java machines as needed. Access to this information requires a security aspect.

Logical Threads: As the FMA is intended to support active, autonomous, management applications, components must be able to support concurrent and re-entrant conditions with respect to threads. Management applications are made of distributed components, so the FMA introduces the concept of a logical thread that spans processes, and in particular, is capable of spanning execution threads in different virtual machines [17]. Thus, component behavior with respect to threads may be specified with respect to *logical* threads instead of just the provided Java language threads.

2.4 The Use of Jini Technology

Jini technology is used within the Jiro technology for a number of purposes. It is used to discover federated Java virtual machines (termed *stations*), which are the active component of the Jiro technology, and supports addressing domains within a federation of stations, as the transaction manager (including *leases*), and for a variety of lookup operations, including discovery of CIM Object Managers, and various other components, interfaces, and services running on those platforms.

2.5 Other Basic Services

The basic services of the FMA include those provided by the Jini environment, plus logging, scheduling [18], and so on. Services are regarded as “basic” within the Jiro technology if they are assumed present on every platform. The criterion for regarding a particular service as being basic is usually the need for it to be pervasively used throughout management applications.

“Services” that are not pervasive may be supplied by components, and may be discovered (see “discovery”, as discussed under “The Use of Jini Technology” above) when needed.

2.6 The use of components for filling out services and functionality

As hinted at in the previous paragraph, a significant use of FederatedBeans components is providing additional functionality and services on a Jiro technology platform, without need to define an extension to that platform. As the use of this platform matures, it is expected there will be a large number of service components supplied by interested parties.

Should the use of any of those services become so frequent as to fulfill the “pervasive” criterion, consideration could be given at that time to re-awakening the community process. to modify the FMA Specification, defining an *extension* to the architecture.

2.7 **Five stakeholders in the value proposition**

Five broad classes of “stakeholder” in the Jiro technology may be identified. A “stakeholder” is a person or user that has something to gain, or lose, by use of the technology. In the following sections, each of these stakeholders is identified, and their “stake” described.

Stakeholder—the Resource Vendor

The resource vendor is the manufacturer of such things as disks, tape drives, storage subsystems, software products, and so on.

Hardware and software vendors offering products in the range of a few tens of dollars to thousands of dollars (US) face strong competitive pressures with thin profit margins. Vendors in this space typically produce 10^5 to 10^7 devices per year at very low cost and profit margin. Example of device retail costs (at this time) include:

Device	***Market Price***
CD-ROM (40x)	$29
8 GB Tape Drive	$59
10 GB Disk Drive	$121
Celeron Computer	Free or $399

Products in this price range are extremely price sensitive as consumers often care little about brand name or quality and will often purchase the lowest price product. Any additional cost to support manageability is unacceptable. Vendors in this arena can reduce distribution costs by providing any software (on floppies or CD’s) already bundled in the box, their support software with management application vendors software, or by Web download only. Vendors can participate in the management arena by simply developing a CIM provider for their device, a once-only development cost.

Vendors producing products on the low-end of cost and profit will benefit from Jiro technology in several ways:

- the vendor can play in the CIM/WBEM world at a very low initial cost and also be managed in the Jiro technology-based world as well. In Microsoft Windows, the OS where most commodity hardware is installed, the vendor develops a CIM provider. It is then supported in the Microsoft Management Console (MMC). This first step allows the device to be managed by exposing all the device’s “knobs.” This allows the device to be managed at a higher level by intelligent FederatedBeans components in concert with other devices and services;
- a large demographic of the customer population needs manageable devices and will be swayed in their purchasing decision by the device’s integration with the FMA. The cost of developing the support can be amortized over a large number of devices and the cost of manufacturing the software for inclusion with each device is low;
- in a two-step development lifecycle, the vendor can “get into the game” with the low entry cost of developing the CIM provider, perhaps giving away the management

software, and later developing the intelligent FederatedBeans components that manage the device in the most efficient and effective manner possible; and,
- for very little investment, the vendor can provide the kind of functionality and manageability that was previously available in devices costing 10× - 100× as much by leveraging the Jiro technology infrastructure.

The benefits for the high-end vendor are very similar, but with the added benefit of the vendor probably wanting to, and being able, to provide a FederatedBeans solution. This may provide special control or understanding of the larger subsystem (including, perhaps, a "contact the support center" function that implements a 24 × 7 maintenance policy that requires no intervention by the customer). The advantages are:
- the CIM technique allows the description of sub-systems of arbitrary complexity;
- the FederatedBeans components approach allows a vendor to supply system-specific components that may be also utilized in other management products; the manufacturer does not have to develop a "complete" management system in order to enable the one or two essential features needed for the added value of their product; and,
- The FederatedBeans approach can ensure that the sub-system appears on management consoles in a way that the manufacturer wishes (together with that manufacturer's own appearance and message).

Stakeholder—the Component Vendor

Much of the success of Jiro technology will be in the existence of an active market in Jiro components, *i.e.*, software components that can be used as building blocks in the creation of storage management applications. The leverage of this approach is that developers of management applications can recast their work in terms of integration of components, rather than the design, development and testing of every component needed to make a complete application.

In many other parts of the software industry, software components have become a very successful, and necessary, part. Software components can take the form of source and binary libraries, dynamically loadable libraries, shared objects, DCOM [19] components, foundation component sets, as ActiveX components and Java packages and applets.

There are two approaches to the use of components: by those wishing to provide specialized components, such as those described above, where specific and dedicated functionality is provided as a component behind standard interfaces, and by those providing general functionality in components with interfaces that extend the range and capability of the entire system.

Examples of this latter type of component could include:
- a health monitor, that collects the values on certain attributes, and delivers warnings when any of these values move outside predetermined limits;
- directory and lookup services, powered by various difference sources of information, *e.g.*, DNS, or by different access standards, *e.g.*, LDAP;
- an asset manager, that integrates what is installed (visible to the component) with an enterprise inventory system;

- event handlers, that do correlation, in order to deduce the root cause of an event storm, *e.g.*, failure events from many routers about not being able to reach certain hosts may all be due to a power failure on just one segment of a network;
- time series analysis, given various observations of some measure at known time intervals, so that future values may be predicted;
- virtual volume tuner, that uses performance statistics from the virtual volumes to adjust the behavior of real disks and their interconnections to improve the performance of the virtual volumes;
- a database configurator, that given a set of parameters about the intended use of a database, can configure virtual volumes, table layouts, and other controls, in order to either enable the intended database use, or to optimize behavior;
- a capacity planner, which not only can assess what is installed, but may also be able to reach product information on manufacturer's web sites, so that an upgrade plan can be derived by playing "what if";
- a storage area network tool, that analyzes a topography of a traffic pattern, and advises on the addition or movement of existing network access points in order to balance use of the network segments; and so on.

The "value" of components may be in their intrinsic value, and thus be traded and sold "off the shelf", like many other applications, or in the value that they enable in other equipment (so-called "drag"), where the software is essentially given away, in order to improve sales of the equipment.

Stakeholder—the Management Application Vendor

For those developing management application, the advantages are:

- that FMA enables developers to build applications with advanced, automated functions that realize the goal of managing storage or storage networks, where many of those automated functions may be obtained "off the shelf";
- that FMA provides the FederatedBeans model that enables interoperability among diverse applications, services, and devices, and is also an aid in the architecture and design phases of the application;
- relieving the application developer of the necessity to design and implement the means of accessing the management information of devices, and for these to be easily added, removed, or provisioned for service; and,
- reducing downtime by enabling automatic updates to applications or services.

If a new application or device is Jiro technology-enabled *and* a management component vendor has a FederatedBeans product implemented, the new application or device will be immediately capable of being incorporated into the management environment. For most devices, being WBEM-enabled will be sufficient for Jiro enablement. From the customer's point of view, the new equipment will be capable of being managed (at a higher rather than lower level) and reported a standard manner. A Jiro-enabled disk array, for example, would be able to report capacity, which could then be used by applications such as capacity planners. The fact that the disk array is there and has been recognized means that volume managers can take advantage of it automatically—rather than having to be told its whereabouts.

Stakeholder—the Information / Data Application Vendor
Information and data application vendors can improve the performance of their applications by being able to interact directly with the management components of the data storage system. For example, if a the data application is a backup suite, the implementation of that suite could:

- use the management system to discover which files need backing up, without having to directly use the file system interfaces; this reduces porting costs in development;
- use a propriety interface to set up the backup application (it would be possible to develop a CIM interface, but an intermediate solution would be to use existing CLI's *via* a specially developed facade);
- similarly, proprietary interfaces might be used to collect information, say from a log file, which could be used as key for the generation of events that a FederatedBeans component could use to report upon the status of the backup; and,
- since the CIM model includes objects for the management of tape libraries, the management of the tape pools could be integrated with the backup application to ensure correct rotation of tapes, and the observation of the correct policy rules for the keeping of tapes in the rotation.

Stakeholder—the Customer
The CIO: the Jiro technology, the FederatedBeans components, and the basic services define a baseline against which management and data applications can be measured. The CIO is assured that an Jiro-enabled product meets basic requirements for interoperability with other applications. Further certainty may be obtained by insisting on management applications that have been Jiro *certified*, and by ensuring that the producer has participated in interoperability tests with other products.

Over a period, by invoking careful acquisition policies, a CIO and the IT Department may build a more fully integrated set of management capabilities by looking for Jiro-compliant applications.

The System Administrator: From the point of view of the system administrator, the acquisition of Jiro-based applications, and value-based FederatedBeans components:

- minimizes barriers for providing management of many hardware/OS platforms; eliminates or minimizes platform porting, enables solution developers to support platforms that may have a lower priority in the company's target market;
- provides broad device support: any device with a WBEM provider or supporting SNMP can be managed through a FederatedBeans component; and the support of private interfaces allows management of non-WBEM and non-SNMP devices;
- enables a finer application granularity: components allow users to pick and choose Jiro-enabled applications and 3rd party FederatedBeans solutions rather than others;
- brings management capability from a storage-specific console or an enterprise management console: The Jiro 3-tier architecture separates management logic from user interface and avoids mandating particular user interface solutions.
- developing through component design and assembly allow the user to take more ownership of policy and automation.

3 Related Work

The following are possible choices, among others, to implement a second tier in a management system:

- *Enterprise JavaBeans*™ (EJB™) [20] component architecture is designed to be the most capable technology for second tier "business logic," providing single threads of logic execution that normally originate in the client tier, and transactions that are usually with respect to a single third tier database. The FederatedBeans model is suited to the creation of management solutions as it more naturally supports thread concurrency, makes specific provision for the support of a CIM-based third tier; and has the ability to support arbitrary transactions with respect to that third tier;
- WBEM, which is the preferred choice for the third tier, could also to provide an object model and schema for the second tier. Further work would be required, as the CIM Schema would have to be endowed with the appropriate new objects or extensions. Even then, it would remain a "double technology" for implementation, *i.e.*, one technology for the definition of the objects, and another (platform dependent) for the definition of methods. To be completely capable for utilization in the second tier, WBEM would also need to be given a component model that addresses the same issues as listed for the FederatedBeans model;
- CORBA [21] appears to be an appropriate choice, but has had limited use in the implementation of management applications. Its success in business logic does not argue for success in the management arena. CORBA objects are still platform-specific, thus creating a "porting" problem, even though there are no impediments to inter-platform communication.

It must be feasible to choose the first and second tiers independently. In order for there be a choice, the second and third tier technologies must decoupled by an appropriate choice of interfaces between the tiers. Jiro technology appears to be the appropriate choice.

4 Future Work

By the time this paper is published, a reference implementation of the Jiro technology, implementing the FMA, should be publicly available. It is also intended that by that time a number of the original Expert Group participants will have also applied the FederatedBeans concept to the production of a useful number of components, that the interoperability of these components will have been demonstrated, and their usefulness in creating management solutions be in the course of evaluation.

As further FederatedBeans components are developed, some will be found pervasive enough to be "basic" in the sense of the FMA Specification. At that time, the Expert Group could reconvene to integrate candidates into revisions of the FMA Specification.

5 Conclusions

The development of an accepted and useful architecture for the building of management applications marks an important turning point in the arena of storage management. The FMA, and Jiro technology represent both merging and emerging developments making further advance possible. The realization of this architecture in real products is the next major objective.

Thanks
The authors wish to thank, particularly, the members of the FMA Expert Group for the efforts that went into creating, editing, and revising the FMA Specification. It is from this foundation that the Jiro technology has developed into a viable storage management solution. The authors also want to thank the session chair for the efforts he has marshaled, including the anonymous reviewers, to correct and revise this paper. As always, any final errors and omissions are the responsibility of the authors.

References

[1] Shiers, Jamie: "Massive-Scale Data Management using Standards-Based Solutions," ***Proc. 16th IEEE Symp. Mass Storage***, San Diego, CA, IEEE Computer Society Press (March, 1999).
[2] Coyne, R.A.; and Hulen, H.: "An Introduction to the Mass Storage System Reference Model, Version 5," ***Proc. 12th IEEE Symp. Mass Storage***, Monterey, CA, IEEE Computer Society Press (April, 1993).
[3] Gosling, James; Joy, Bill; and Steele, Guy: **The *Java™* Language Specification.** Addison-Wesley, Reading, Massachusetts. ISBN 0-201-63451-1 (1996)
[4] Edwards, W. Keith: **Core *Jini™*.** Prentice Hall PTR, Upper Saddle River, New Jersey. ISBN 0-13-0114469-X (1999).
[5] FMA Expert Group: **Federated Management Architecture Specification, Draft Version 1.0.** Sun Microsystems, Inc., http://www.jiro.org/specs.html. (January, 2000).
[6] Sun Microsystems, Inc: **The *Java™* Community Process.** Sun Microsystems, Inc., http://java.sun.com/aboutJava/communityprocess/. (May, 1999).
[7] IEEE Storage Systems Standards Working Group: **Mass Storage System Reference Model, Version 5.** http://ssswg.org/public_documents/MSSRM/V5toc.html (September, 1995).
[8] Technical Development Committee: **The CIM Specification, Version 2.2.** http://www.dmtf.org/spec/cims.html, Distributed Management Task Force, (June, 1999).
[9] DMTF XML Working Group: **CIM XML Mapping, Version 2.0.** http://www.dmtf.org/XML/CIM_XML_Mapping20.htm, Distributed Management Task Force, (June, 1999).
[10] DMTF XML Working Group: **The CIM HTTP Mapping, Version 1.0.** http://www.dmtf.org/XML/CIM_HTTP_Mapping10.htm, Distributed Management Task Force, (June, 1999).
[11] Technical Development Committee: **The WBEM Initiative.** http://www.dmtf.org/wbem/, Distributed Management Task Force, (1999).
[12] Englander, Robert**: Developing Java Beans.** O'Reilly and Associates, California. ISBN 1-56592-289-1 (1997).
[13] Gamma, Erich; Helm, Richard; Johnson, Ralph; and Vlissides, John: **Design Patterns: Elements of Reusable Object-Oriented Software.** Addison-Wesley, Reading, Massachusetts. ISBN 0-201-63361-2 (1994).
[14] Lindholm, Tim; and Yellin, Frank: **The *Java™* Virtual Machine Specification.** Addison-Wesley, Reading, Massachusetts. ISBN 0-201-63452-X (1996).
[15] Bernstein, Philip A.; and Newcomer, Eric: **Transaction Processing.** Morgan Kauffman Publisher, Inc, San Francisco, California. ISBN 1-55860-415-4 (1997).

[16] Oaks, Scott: ***Java™* Security.** O'Reilly and Associates, Sebastopol, California. ISBN 1-56592-403-7 (1998).
[17] Haddon, Bruce K. and Connor, William H.: "Software for Distributed Monitor Concurrency Control," Patent Pending, U.S. Patent and Trademark Office (November, 1998)
[18] Haddon, Bruce K.: **Machine-Independent Real-time Operating System Interfaces**. Ph.D. Thesis, Department of Computer and Electrical Engineering, University of Colorado, Boulder (1979).
[19] Eddon, Guy; and Eddon, Henry: **Inside Distributed COM.** Microsoft Press, Washington. ISBN 1-57321-849-X (1998).
[20] Valesky, Thomas B.: **Enterprise JavaBeans™.** Addison Wesley Longman, Inc., Massachusetts. ISBN 0-201-60446-9 (1999).
[21] Joint Revised Submission: **CORBA Components.** Object Management Group, Inc. (1999)