



**Openwings
Management Services Specification
Ver 1.0 Final**

COPYRIGHT ©2000-2003 General Dynamics Decision Systems, INC.
ALL RIGHTS RESERVED
This document is subject to "Terms of Use" as described at <http://www.openwings.org>.

PREFACE

About the Open Specification Process

A brief overview of the community process for developing Openwings Specifications is discussed below. This process was developed by General Dynamics Decision Systems and is similar to the community process for developing Java specifications.

A Process Management Organization (PMO) oversees development and maintenance of all Openwings Specifications. Any company, organization or individual can join the Specification development team; team members are called Participants. The latest Openwings Specifications will be maintained on a Public Web Site.

Participants can submit requests to the PMO to modify existing or develop new Specifications. If the PMO accepts the request, the PMO will then begin the process of forming an Expert Team and will appoint a Specification Lead.

The Specification Lead will direct the Expert Team in developing the new Specification. Once the team agrees on a draft of the Specification, it will be posted on the Public Web Site for public review. A member of the Expert Team will also begin development of a Reference Implementation and Compatibility Test Suite for the new Specification.

Based on public comments, a final release of the Specification will be produced by the Expert Team. Concurrently, the Reference Implementation and Compatibility Test Suite will be completed. Once the Final Public Release of the Specification is completed and posted on the Public Web Site, the Expert Team will disband.

This process is intended to provide rich consensus based Specifications.

Contributors

We would like to recognize and thank the following people for their contributions to this document.

Author(s):

Michael Smith, Guy Bieber, Jeffrey Carpenter

Expert Team Members:

Stuart Lewin, BAE Systems North America

Ramesh Nagappan, Sun Microsystems

Ayal Spitz, Mitre Corp

Dr. Dave Usechak, Ocean Systems Engineering Corporation

1.	INTRODUCTION	1
1.1	PURPOSE	1
1.2	SCOPE	1
1.3	DEFINITIONS	1
1.4	OVERVIEW	2
1.4.1	<i>Openwings Overview</i>	2
1.4.2	<i>Document Overview</i>	2
2.	GOALS / REQUIREMENTS.....	4
3.	USE CASES.....	5
3.1	CREATE MBEAN	5
3.2	CREATE MBEAN USER INTERFACE.....	5
3.3	DEPLOY MBEAN	5
3.4	DEPLOY MBEAN USER INTERFACE	6
3.5	MANAGE	6
3.6	ADAPT MBEAN	6
4.	ARCHITECTURE.....	7
4.1	BACKGROUND.....	7
4.1.1	<i>Federated Management Architecture (FMA)</i>	7
4.1.2	<i>Java Management Extensions (JMX)</i>	9
4.1.3	<i>Web Based Enterprise Management (WBEM)</i>	11
4.1.4	<i>Simple Network Management Protocol (SNMP)</i>	13
4.2	OPENWINGS MANAGEMENT FRAMEWORK	14
4.3	MBEAN	16
4.4	MBEAN USER INTERFACES	16
4.4.1	<i>Openwings Management Bean Adapter Model</i>	16
4.4.2	<i>Policies and Management</i>	16
4.4.3	<i>Bridging with JDMK</i>	16
4.5	DEPLOYMENT.....	17
5.	INTERFACES.....	18
5.1	NET.OPENWINGS.MANAGEMENT.MBEAN	19
5.1.1	<i>getMBeanInfo</i>	19
5.2	NET.OPENWINGS.MANAGEMENT.REMOTEMANAGER.....	20
5.2.1	<i>getInetAddress</i>	20
5.2.2	<i>getAdapterURLList</i>	21
5.2.3	<i>getMBean</i>	21
5.2.4	<i>getAllMBeans</i>	21
5.2.5	<i>addManagerListener</i>	21
5.2.6	<i>removeManagerListener</i>	22
5.2.7	<i>getUserInterfaceFactories</i>	22
5.2.8	<i>getUserInterfaceFactories</i>	22
5.3	NET.OPENWINGS.MANAGEMENT.MANAGERLISTENER.....	24
5.3.1	<i>processMBeanAdded</i>	24
5.3.2	<i>processMBeanRemoved</i>	24
5.4	NET.OPENWINGS.MANAGEMENT.MANAGER	25
5.4.1	<i>addMBean</i>	25
5.4.2	<i>removeMBean</i>	25
5.4.3	<i>addMBeanAdapter</i>	26
5.4.4	<i>removeMBeanAdapter</i>	26

5.4.5	<i>addUserInterfaceFactory</i>	26
5.4.6	<i>removeUserInterfaceFactory</i>	26
5.4.7	<i>shutdown</i>	26
5.5	NET.OPENWINGS.MANAGEMENT.MANAGERFACTORY	28
5.5.1	<i>getManager</i>	28
5.6	NET.OPENWINGS.MANAGEMENT.MBEANADAPTER	29
5.6.1	<i>getAdapterURL</i>	29
5.6.2	<i>shutdown</i>	29
5.7	NET.OPENWINGS.MANAGEMENT.MANAGEMENTPOLICY	30
5.7.1	<i>setMBeanClasses</i>	30
5.7.2	<i>getMBeanClasses</i>	30
5.7.3	<i>setMBeanAdapterClasses</i>	31
5.7.4	<i>getMBeanAdapterClasses</i>	31
5.7.5	<i>setUserInterfaceFactories</i>	31
5.7.6	<i>getUserInterfaceFactories</i>	31
6.	TOOLS / UTILITIES	32
6.1	OPENWINGS SHELL/EXPLORER.....	32
7.	EXAMPLES	34
7.1	CREATING AN MBEAN	34
7.2	CREATING AN MBEANADAPTER	34
8.	COMPLIANCE.....	36
9.	REFERENCES AND FURTHER READING.....	37

FIGURE 1:	DEFINITIONS.....	2
FIGURE 2:	GOALS / REQUIREMENTS TABLE.....	4
FIGURE 3:	USE CASES.	5
FIGURE 4:	FMA ARCHITECTURE.	8
FIGURE 5:	JMX MANAGEMENT MODEL.....	10
FIGURE 6:	JDMK MANAGEMENT.....	11
FIGURE 7:	THE WBEM TRIANGLE.....	12
FIGURE 8:	BROWSER-BASED ACCESS TO A WBEM-COMPLIANT SYSTEM.	13
FIGURE 9:	OPENWINGS MANAGEMENT ARCHITECTURE.	15
FIGURE 10:	JDMK INTEGRATION.....	17
FIGURE 11:	INTERFACE ROLES.	18
FIGURE 12:	MANAGEMENT INTERFACES.	18
FIGURE 13:	MANAGEMENT BEANS IN OPENWINGS EXPLORER.....	32
FIGURE 14:	EXAMPLE MBEAN USER INTERFACE.....	33
FIGURE 15:	SAMPLE MBEAN CODE.....	34
FIGURE 16:	SAMPLE MBEANADAPTER CODE.....	35
FIGURE 17:	COMPLIANCE TABLE.....	36

1. Introduction

1.1 Purpose

This document defines the Openwings Management framework. The purpose of this framework is to provide management of Openwings components and services. Since one of the major goals of Openwings is the ability to build systems that require zero-administration at run-time, most of the management services are maintenance purposes or to configure policies prior to deploying the system.

1.2 Scope

This document focuses on the management framework necessary to automatically manage system operations including security, system formation and services. These examples only serve as use cases for this document. This document requires an understanding of the Openwings Components and the Openwings Policies. Familiarity with the Java Dynamic Management Kit (JDMK) is helpful.

1.3 Definitions

Term	Definition
Client	The client communicates with the management interface of the resource being managed.
Common Information Model (CIM)	CIM is an implementation-neutral schema for describing management information.
Distributed Management Task Force (DMTF)	DMTF is an industry organization leading the development, adoption, and unification of management standards and initiatives for desktop, enterprise and Internet environments.
Federated Management Architecture (FMA)	FMA describes a three-tiered architecture for management applications. The three tiers are resources, management services, and clients. In FMA, clients are hosted by Java Virtual Machines (JVMs), management services by JVMs enabled as management servers, and resources by any appropriate host machine, including a JVM.
Java Dynamic Management Kit (JDMK)	JDMK is a commercial management tool from Sun Microsystems that is JMX compliant. The features of JMX were determined, in part, based on experience with early versions of JDMK.
Java Management Extensions (JMX)	JMX defines architecture, design patterns, services and an API for application and network management.
Management Bean (MBean)	An MBean is a lightweight management plug-in attached to components. Typically a single MBean is used to manage a single aspect of a component. Thus, a component may have multiple MBeans.
Resource	A system resource to be managed. In general, managed resources may include network appliances, devices, and applications. In the context of this document, resources refer to Openwings components, their policies and their services.
Simple Network Management Protocol (SNMP)	SNMP is a message-based protocol that has become the de facto standard for managing network devices.
Context	A context is a grouping of components. Contexts may be used to group

	services based on network topology or security, for example. Contexts are fully defined in the self-forming systems specification.
User Interface Factory	A User Interface Factory is an object that can create user interfaces for specific types of management beans
Web-Based Enterprise Management (WBEM)	A set of management and Internet standard technologies developed to unify the management of enterprise computing environments. It provides the ability for the industry to deliver a well-integrated set of standards-based management tools, leveraging ubiquitous technologies such as CIM, XML and HTTP.

Figure 1: Definitions

1.4 Overview

1.4.1 Openwings Overview

Openwings™ is an open community, non-proprietary effort to define specifications for self-forming, self-healing systems. Motorola (now General Dynamics Decision Systems) and Sun Microsystems established the Openwings™ consortium in June of 1999. Since then, over 100 companies have registered to help mature its development, and more companies are registering daily.

The core Openwings™ framework is designed to incorporate existing commercial standards and is intended for use in both commercial and military environments. Openwings™ is being developed using a community development process modeled after the very successful Java Community Process. Anyone may join the Openwings™ community, participate on expert teams, or use the resulting specifications free of charge by merely signing up at the community web site (<http://www.openwings.org>).

The Openwings™ Architecture provides a framework for building plug-and-play, service-oriented, network-centric, self-forming, self-healing systems that are independent of middleware, databases, platforms, and deployment contexts. Openwings™ has a special focus on issues of availability, security, and interoperability. Openwings™ is the embodiment of a new movement in the software engineering community towards a paradigm known as Service-Oriented Programming (SOP). For an introduction to SOP, refer to <http://www.openwings.org/download/specs/ServiceOrientedIntroduction.pdf>.

1.4.2 Document Overview

The primary building block in Openwings is the component. Components are self-contained software elements that provide application services in the Openwings system. Every Openwings component shall be manageable. The Openwings Management framework provides a standard way to add management plug-ins, called Management Beans (MBeans), to manage different aspects of the component. These management beans allow operations to be performed at runtime on components that are outside the components' normal operating interfaces. For a maintainer this can be used to turn on and off debug logging. For a security manager this would allow various policies to be modified.

This specification follows the basic format of all of the Openwings specifications:

- Goals and Requirements – Describes requirements of this specification.
- Use Cases – Describes UML use cases for this specification.
- Architecture – Describes the various architectural aspects of this specification.
- Interfaces – Describes interfaces required for this specification.
- Tools – Describes various tools and utilities provided with the specification.
- Examples – Provides representative real world use cases for this specification.
- Compliance – This section describes what is required to comply with this specification.

2. Goals / Requirements

Here are the goals / requirement for this specification. Each requirement references one or more of the top-level requirements found in the Openwings Overview Specification.

#	Goals / Requirements	Ref
1.	Shall provide automated management, i.e., zero admin.	8
2.	Shall provide an open framework for adding management modules (MBeans).	7
3.	Shall provide distributed remote management.	4
4.	Shall provide highly available management.	9
5.	Shall provide a management interface through HTML.	27
6.	Shall provide a management framework that allows plugins of different management user interfaces.	7
7.	Shall provide a method for integrating with other management protocols including FMA, SNMP, and JMX.	27
8.	Shall use the Openwings Connector framework for communications.	6

Figure 2: Goals / Requirements Table.

3. Use Cases

The following figure shows the use cases for Openwings Management.

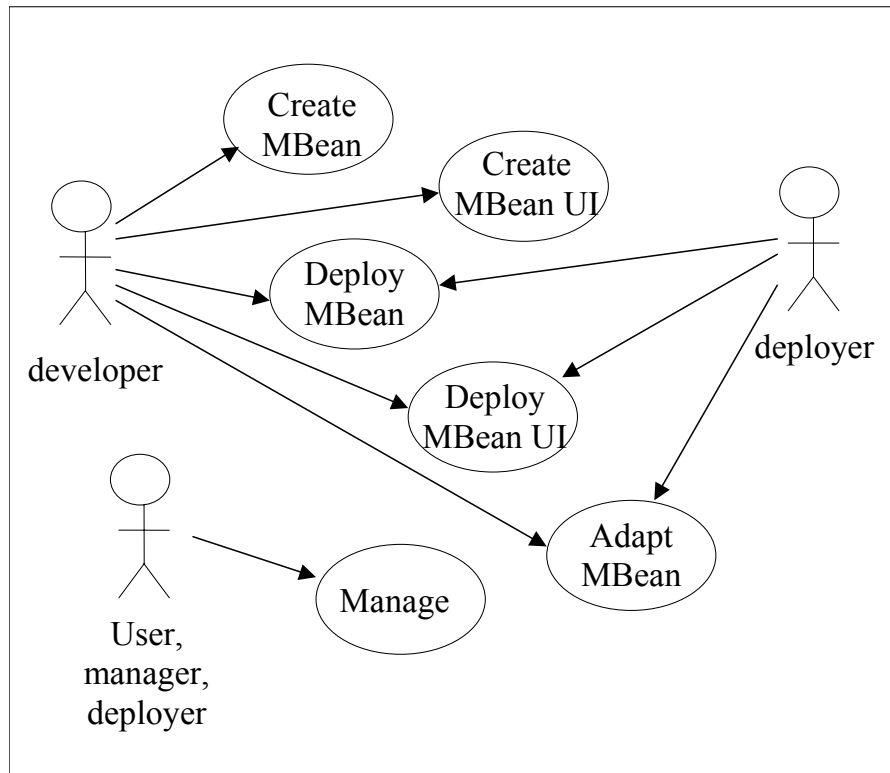


Figure 3: Use Cases.

3.1 Create MBean

The Systems Engineer participates in the creation of an MBean by defining the requirements for the MBean. The Developer participates in the creation of the MBean by creating the MBean itself.

3.2 Create MBean User Interface

The Systems Engineer participates in the creation of a MBean User Interface by defining the requirements for the user interface. The Developer participates in the creation of the user interface by creating the user interface factory and associated classes.

3.3 Deploy MBean

The Developer or the Deployer deploys the MBean into the system by packaging it in a way that enables the system to access it, and by configuring the system to use the MBean. Configuration of the system will likely involve the configuration of one or more policies.

3.4 Deploy MBean User Interface

The Developer or the Deployer deploys the MBean User Interface into the system by packaging it in a way that enables management clients to access it.

3.5 Manage

Once an MBean and its User Interface have been deployed into the system, the system can be managed. Actors that manage the system are the User, Manager, and Maintainer. Sample management tasks include the configuration of policies and the observation of built in test (BIT) data.

3.6 Adapt MBean

The developer can create MBeanAdapters that make MBeans accessible outside of a component. The deployer of each component can attach these adapters to components to make MBeans in these components available in the context of some other management framework.

4. Architecture

4.1 Background

Challenges to managing distributed, dynamic, service-based systems include the discovery of resources and management logic; providing communication between clients, managers and resources; and ensuring security in the management process. There are a handful of technologies that provide relevant background for the discussion of distributed systems management. The Federated Management Architecture (FMA) and Java Management Extensions (JMX) specification are emerging management technologies that are specific to Java. Web Based Enterprise Management (WBEM) is an industry initiative to standardize management in a language-neutral manner. WBEM has considerable industry support and deserves consideration for any distributed management system. Simple Network Management Protocol (SNMP) is the current de facto standard for network management.

4.1.1 Federated Management Architecture (FMA)¹

The Federated Management Architecture (FMA) is an n-tiered architecture for distributed management applications. Common management solutions employ three tiers: clients, management services, and resources. This configuration is illustrated in the following figure.

¹ Based on the Federated Management Architecture (FMA) Specification v1.0 rev 0.4, January 21, 2000.

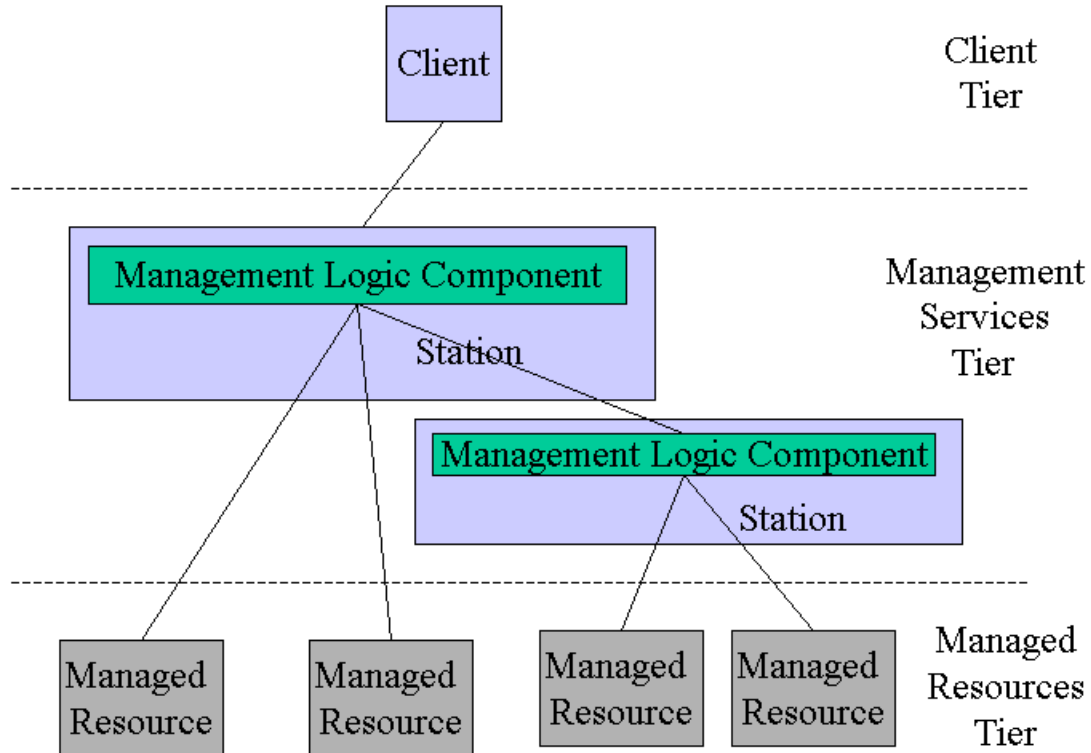


Figure 4: FMA architecture.

The Client Tier exists for the purpose of gathering and displaying data. Typically, the client is a management console of some kind. Clients connect to the Management Services in the middle tier via some form of inter-process communication.

The Management Services Tier acts as the gateway to the resources being managed. This tier contains the management logic that is required to control the resources. The management logic may exist in a single component or in multiple components. FMA management servers are Jini services; they are also called Stations. A Station corresponds to a JVM that supports Jini. Management Services exist within the context of a Station. In Openwings, this is analogous to the Component Services model. Stations register themselves with the Jini lookup service for their management domain. The management domain is simply a logical grouping of services similar to Openwings Contexts. Stations, through Jini, provide a framework for federating distributed management services.

The Managed Resources Tier consists of the resources being managed by the Management Services Tier. These resources may be physical devices or software. The means of communication between the management services and the resources is determined by the communication abilities of the resource.

FMA has developed an abstraction over Java's remote method invocation (RMI) paradigm to facilitate seamless communications between local and remote objects. The model is proxy-based and uses the façade design pattern to provide the client and management services with a uniform view of the resource being managed. This technique is essentially the same as RMI connectors in the Openwings Connector model. However, Openwings Connectors have the advantage of being based on Java interfaces.

FMA was started with the intent of defining a specification for management of resources in service-based architectures, Jini in particular. In order to fully define the management architecture, the specification team found it necessary to define several supplementary technologies including logging, transactions, persistence, and events. With the possible exception of persistence, each of these supplementary technologies is defined or is being defined separately through the Java Community Process. Unfortunately, the FMA specification is complicated with the supplementary technologies needed to make FMA work. The lone exception is security. FMA adopts and extends the Java Authentication and Authorization Service specification to implement security.

4.1.2 Java Management Extensions (JMX)²

The Java Dynamic Management Kit (JDMK) is a commercial management framework that Sun Microsystems has been selling for several years now. Based in part on this work, the Java Management Extensions (JMX) initiative has the objective of defining standard architecture, design patterns, services and an API for application and network management. At this time, JDMK is an implementation of JMX while JMX and FMA are completely separate management models. That is, there is no direct support for FMA in JDMK and vice-versa.

Figure 4 depicts the JMX management model. JMX architecture consists of three levels: The Instrumentation Level, the Agent Level and the Manager Level. The Instrumentation Level is the most basic level. At this level, Management Beans (MBeans) are employed to instrument resources being managed. Simply put, an MBean provides a programmatic interface for managing a system resource (e.g., disk drive, application). MBeans come in two flavors: Static and dynamic. Dynamic MBeans can change their behavior (including fields and method signatures) at run time.

The JMX Agent Level defines the concept of an MBean server. The MBean server is used to register and interact with MBeans. At this time, there is no direct access to an MBean; the MBean server mediates all interactions. There is one MBean server per JVM. Also defined at the Agent Level are protocol adaptors that enable external applications to interact with the MBean server through various protocols (e.g., SNMP, HTTP). Protocol adaptors perform essentially the same function as Openwings Connectors. JMX does not define an API for protocol adaptors, but they must be implemented as MBeans in order to register with the MBean server.

² Based on the Java Management Extensions Instrumentation and Agent Specification v1.0, December 1999.

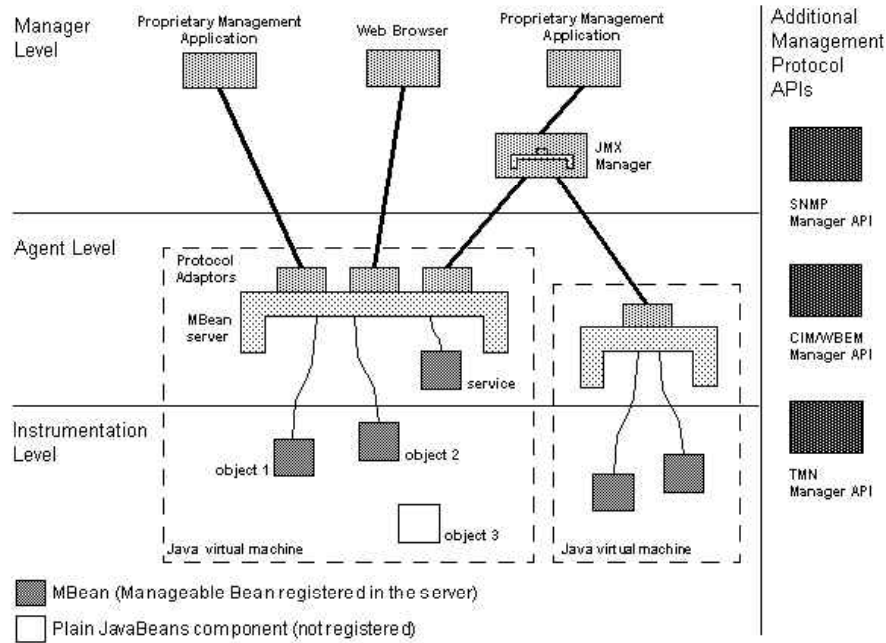


Figure 5: JMX Management Model.

The JMX Manager Level is the domain of custom management applications. These might include IBM’s Tivoli, HP’s OpenView, or simple web browser applications using CIM/WBEM. Support for different management application protocols is achieved at the Agent Level using protocol adaptors.

JMX defines additional APIs in supplementary specifications for some common management protocols. These are shown on the right side of Figure 4. Using these APIs, developers can build capabilities at the JMX Manager Level and Agent Level to bridge to existing management systems.

4.1.2.1 JDMK

JDMK implements JMX and provides a set of features for remote management using a variety of protocols. JDMK provides support for remote access for management applications via HTML, SNMP, RMI and HTTP. Support for IIOP and CIM are planned. The following figure illustrates the concepts associated with remote management in JDMK.

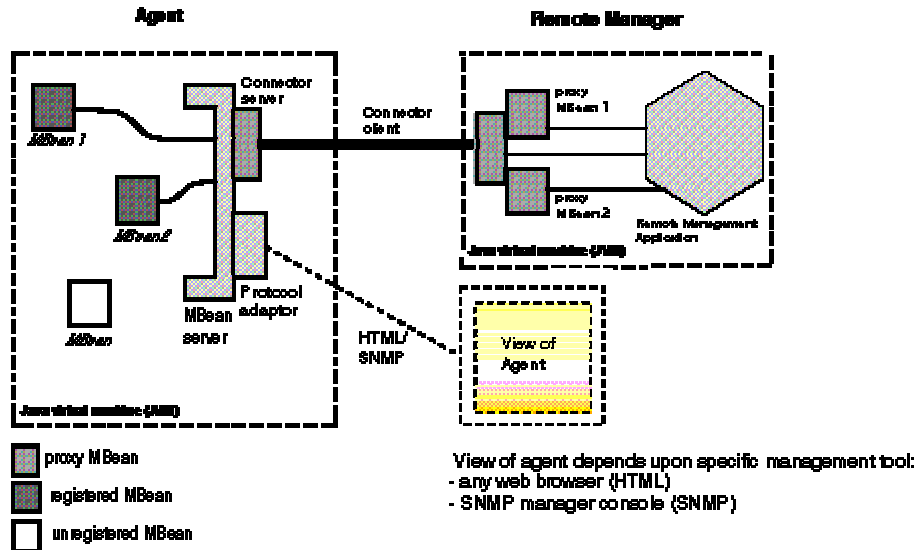


Figure 6: JDMK Management.

While the JDMK product team has indicated that they have prototyped support for Jini services into JDMK, plans to include it in the product are unknown at this time.

4.1.3 Web Based Enterprise Management (WBEM)

Web Based Enterprise Management (WBEM) is an initiative of the Distributed Management Task Force (DMTF) that has the objective of defining a set of management and Internet standard technologies that will unify the management of enterprise computing environments. The intended result is a well-integrated set of standards-based management tools, leveraging ubiquitous technologies such as the Common Information Model (CIM), the eXtensible Markup Language (XML), and the Hyper Text Transfer Protocol (HTTP). The relationship between these three technologies is illustrated in the WBEM triangle shown in the following figure.

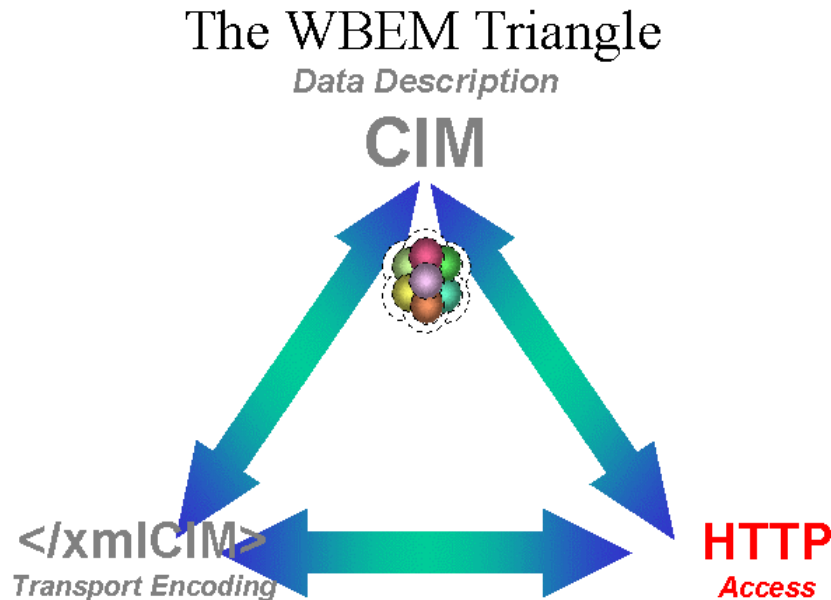


Figure 7: The WBEM Triangle.

CIM is an implementation-neutral schema for describing management information. This facilitates the sharing of management data across different management systems and the integration of management information from different sources. CIM is an object-oriented data model that uses the concepts of classes and instances; CIM is not an implementation. The CIM model provides support for describing system resources and system management in a known, consistent manner.

The XML/CIM language defines a way to represent CIM information in XML documents. This brings the benefits of XML (self-describing, platform-neutral, industry standard) to CIM. With the semantic definitions of interactions and a CIM Object Manager based upon the messages encoded in XML/CIM, applications can reliably interoperate.

The CIM HTTP mapping specification defines the manner in which HTTP is used to transport the CIM information. Examples of CIM HTTP operations include the creation, retrieval, and deletion of classes and instances; retrieval of properties; and method execution. The CIM HTTP mapping implies that management can be done from a web browser as illustrated in the following figure.

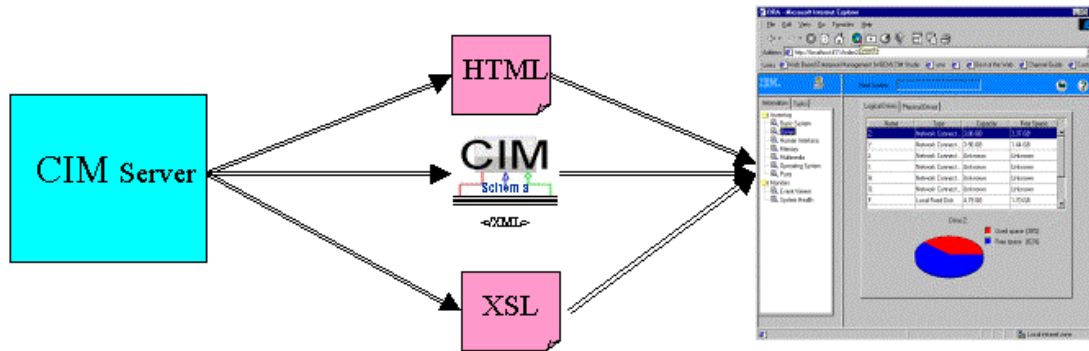


Figure 8: Browser-based access to a WBEM-compliant system.

4.1.3.1 Directory Enabled Network (DEN)

DEN is an initiative to develop a standard and extensible directory schema that enables vendors to provide interoperable network services. The schema is designed to enable interoperable directory services on heterogeneous networks. By adopting a uniform directory representation, applications can transparently leverage network infrastructure on behalf of the user and enable end-to-end services. For example, when a subscriber of an ISP asks for a service, that service must be delivered in an end-to-end fashion. This implies that the data of that service will span multiple vendors' network devices. DEN is a template for exchanging information that enables each vendor to have a common definition of the service but implement that service in their own way (thereby providing added value). The DEN schema is built upon CIM.

4.1.3.2 Desktop Management Interface (DMI)

DMI is a standard framework for managing and tracking components in a desktop PC, notebook or server. It is essentially a registry. DMI is currently the only desktop standard in adoption for today's PCs. The DMI Home Page is a repository of DMI-related information from the specification to tools to support to the Product Registry of DMI-certified products.

4.1.4 Simple Network Management Protocol (SNMP)

Simple Network Management Protocol (SNMP) is the most widely accepted network management protocol on TCP/IP networks. SNMP was designed in the mid-1980s to facilitate communications between different types of networks. It works by exchanging messages called protocol data units (PDUs). SNMP PDUs are essentially objects containing a set of variables. Each variable consists of a variable name, a data type (e.g., integer, string), a value, and access permissions (read, write). SNMP defines five types of PDUs: two for reading resource data, two for setting resource data, and one for trapping network events such as resource startup or shut down. SNMP agents act as management servers for the resources that they manage. Anecdotal evidence suggests that SNMP was

intended as an interim solution until a more full-featured standard could be defined. However, because of its simplicity, SNMP was adopted as a de facto standard and a replacement has yet to emerge. Indeed, even attempts to define SNMPv2— an evolution of the standard that would address some of the shortcomings described below— failed. In April 1999, the Internet Engineering Task Force (IETF) released Draft Standards for SNMPv3. SNMPv3 attempts to succeed where SNMPv2 failed by wrapping and extending previous versions of SNMP. SNMPv3 carries an SNMPv2 or SNMPv1 PDU as its payload and thus integrates with previous SNMP versions. SNMPv3 can be viewed as SNMPv2 plus security and administration. In particular, SNMPv3 addresses the following shortcomings from previous SNMP versions:

- Authentication: This includes message origin identification and ensuring message integrity.
- Privacy: This ensures confidentiality of the message.
- Authorization and Access Control
- Suitable remote configuration and administration capabilities for these features.

It remains to be seen whether SNMPv3 will be widely adopted. Even the most optimistic scenario of widespread adoption of SNMPv3 leaves some deficiencies with regard to distributed management on ad hoc networks. SNMP is designed for the management of networked devices and it does that fairly well. However, SNMP lacks a discovery mechanism. Typically to find SNMP agents you need to know where they are or use IP scanner technology to find them. In addition, the standard data description mechanism for PDUs is clumsy relative to more advanced efforts such as CIM³. This could add unnecessary complexity to the task of managing distributed applications.

4.2 Openwings Management Framework

Analysis of existing management solutions indicated that FMA's management framework contains a number of features beyond the scope of pure management (e.g., connectors, context models, logging), some of which are redundant with Openwings technology and more complex to implement. JDMK provides a highly robust and adaptable management solution in the form of a shipping product, but it lacks direct support for Jini services. The Openwings management framework defines a set of interfaces for managing components in a way that enables the service functionality of component services to be independent of the management functionality. By making management functionality independent of service functionality, services can be managed by different management solutions and management solutions can be applied to different services. The Openwings management framework is designed with the intention of being able to integrate easily with external management frameworks such as JDMK. Figure 8

³ RFC 2570, Introduction to Version 3 of the Internet-standard Network Management Framework, states that: The separation of the management framework from the data description (e.g., MIBs) in SNMP revisions "was designed to allow the SNMP-based protocol to be replaced without requiring the management information to be redefined or reinstrumented. History has shown that the selection of this architecture was the right decision for the wrong reason -- it turned out that this architecture has eased the transition from SNMPv1 to SNMPv2 and from SNMPv2 to SNMPv3 rather than easing the transition away from management based on the Simple Network Management Protocol".

provides a high-level illustration of the Openwings Management architecture. Components contain a Manager that provides a management interface to other applications. The Manager also serves as a container for the lightweight Management Beans (MBeans). A more detailed view of the Manager shows that it provides a context for the MBeans. Client applications can manage the Component via HTML or applets in a web browser.

Because of its ubiquity and ease of use, all Openwings management solutions must support HTML over HTTP for management. In this way, all components will be capable of being managed through ordinary web browsers. One usage scenario is as follows: Within the general Openwings architecture, it is desirable to obtain an HTML view of Openwings contexts. Selecting a context will link to a listing of the components in the context. Selecting a component will link to views of the component, including the management view. Selecting the management view will link to an interface that allows management of the component. In this way, HTML can be used to provide a simple visual drill-down capability on the Openwings network.

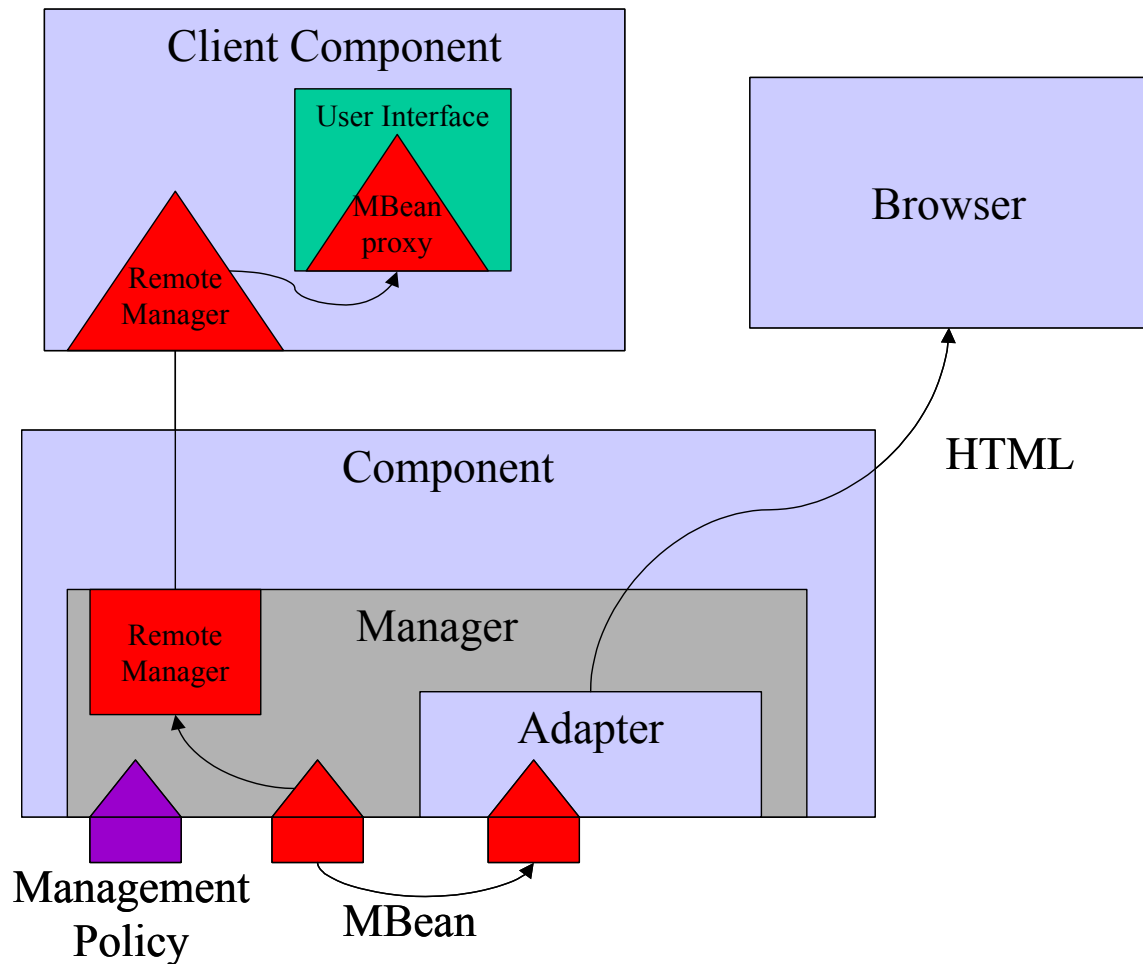


Figure 9: Openwings management architecture.

4.3 MBean

The Openwings management framework uses lightweight management beans (MBeans) that plug into Openwings components to provide management capabilities. These MBeans are analogous to the standard MBeans described by JMX. In short, they are concrete Java classes that implement a unique MBean interface and have a public constructor. MBeans will be accessed through the component's management interface. Openwings connectors will be used to enable MBeans to communicate outside of their components. Openwings connectors provide an interface-based, protocol independent mechanism for communication between components. Connectors can be generated for any interface that contains methods that throw `java.rmi.RemoteException`. Thus, a custom MBean interface can be passed to the connector generator to generate a connector for the MBean.

In the Openwings management framework, a component may be managed by more than one MBean. However, an MBean is designed to manage only one component at a time. MBean User Interfaces may provide an interface to multiple MBeans. When the interface is graphical, different MBeans may be represented on different panels of a tabbed dialog, for example.

4.4 MBean User Interfaces

Depending on the capabilities of the management tool, it may be desirable to obtain a management user interface to use for managing the component. While some user interfaces will be graphical, it is not a requirement. Some may be based on audio or other multimedia. MBean User interfaces are built following the Component Services User Interface Design model. This model separates the functionality of a service from its user interface, coupling them at the point of the service interface. In this way, the service knows nothing of its user interface, allowing it to be reused in a variety of contexts.

4.4.1 Openwings Management Bean Adapter Model

Openwings management provides a way to bridge MBeans into other management frameworks. This is achieved through MBean Adapters. An MBean Adapter is an object which makes an MBean accessible in the context of some other management framework. The adapter will plug into the Manager of a component as shown in Figure 8. The adapter is notified when MBeans are added to a component or removed from a component and takes steps to control the MBean accessibility.

4.4.2 Policies and Management

Openwings policies can be used to automate management tasks. Policies provide an abstraction of context-dependent data and behavior. Because of this, policies can be used to achieve a zero-admin capability and to provide an insertion point for intelligent behaviors.

4.4.3 Bridging with JDMK

As discussed previously, JDMK provides a Telco class management framework that is protocol independent and information independent. Thus, JDMK can provide a bridge to

numerous management protocols including HTML, HTTP, HTTPS, XML, RMI, SNMP, IIOP, and CMIP. Proprietary interfaces can be developed and added dynamically. However, JDMK does not currently provide direct support for Jini services.

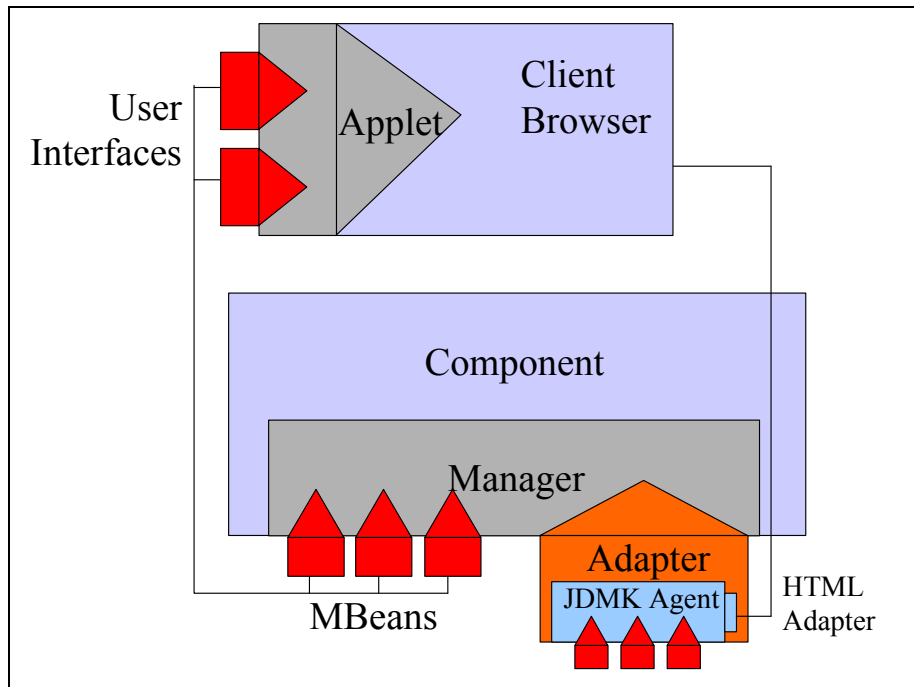


Figure 10: JDMK integration.

The figure above illustrates one way to integrate the Openwings management architecture with JDMK. The JDMK agent can be wrapped in an MBeanAdapter, which is plugged into the Manager object. When Openwings MBeans are added to the Manager, the adapter is notified and adds the MBeans to the JDMK MBeanServer. Once registered with an MBeanServer, the MBeans can be accessed via any of the supported JDMK protocols.

Another solution to integrating JDMK with Openwings is to build a JDMK application that monitors the Jini registry for component registration. As components come on-line, the JDMK application can obtain their management interfaces and register MBeans in JDMK.

4.5 Deployment

MBeans and MBean User Interfaces will be deployed in Java archive (.jar) files. At run time, they can be deployed by their associated applications or by independent applications.

5. Interfaces

The interfaces defined for Openwings management lay out a framework for defining management beans, user interfaces for management beans, generic interfaces for component management, and contexts for adding and removing management beans. The table below lists interfaces and the roles that use them. The role of the User refers to someone who uses an implementation of the management specification; the role of developer refers to someone implementing the specification.

Interface	Role
MBean	User, Developer
RemoteManager	User, Developer
ManagerListener	User, Developer
Manager	User, Developer
ManagerFactory	User, Developer
MBeanAdapter	Developer
ManagementPolicy	Developer, Deployer

Figure 11: Interface Roles.

Here’s how the Openwings management interfaces interrelate: the `RemoteManager` interface describes operations supported by the manager to remote users who wish to access the manager as a service. The `Manager` interface extends `RemoteManager` and provides additional functionality that is used locally to access the `Manager`. The `Manager` contains the `MBeans` for the component. Management applications can obtain `MBeans` from the `Management` interface. The `MBean` shall have no dependencies on its user interface, nor shall the component depend on its `MBean`.

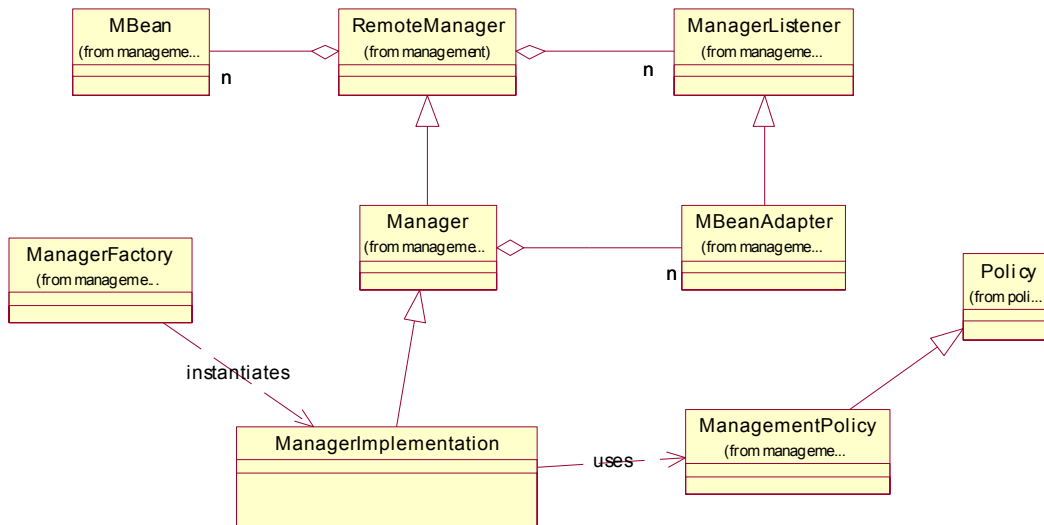


Figure 12: Management Interfaces.

5.1 net.openwings.management.MBean

public interface **MBean**

This is the top-level interface for management beans in the Openwings Management Services framework.

Method Summary

java.beans.BeanInfo	getMBeanInfo() This method provides information about the bean including the name and description.
---------------------	--

5.1.1 getMBeanInfo

public java.beans.BeanInfo **getMBeanInfo()**
throws RemoteException

This method provides information about the bean including the name and description.

Throws:

RemoteException - For any communication-related exception.

5.2 net.openwings.management.RemoteManager

public interface **RemoteManager**

This interface describes a service which can be used to control the management of a component. An implementation of Openwings Management Services should use Component Services to provide a service implementing this interface. This interface provides access to the MBeans installed in a component and the ability to register ManagerListeners for notification of the addition or removal of MBeans. This interface contains a subset of the methods in the Manager interface.

Method Summary	
void	addManagerListener (ManagerListener listener) This method registers an ManagerListener which will be notified when MBeans are added or removed from this Manager.
URL []	getAdapterURLList () This method is used to obtain a set of URLs which can be used to access adapted MBeans (i.e.
MBean []	getAllMBeans () This method returns all MBeans registered with the Manager.
InetAddress	getInetAddress () This method returns the host on which the Manager is running.
MBean []	getMBean (Class componentType) This method will return all MBeans of the requested type that are registered with the Manager.
UserInterfaceFactory []	getUserInterfaceFactories (MBean bean) This method is used to obtain user interface factories available that can build a user interface for a particular service interface.
UserInterfaceFactory []	getUserInterfaceFactories (MBean bean, Class[] desiredUserInterfaceClasses) This method is used to obtain user interface factories available that can build a user interface for a particular service interface.
void	removeManagerListener (ManagerListener listener) This method unregisters a previously registered ManagerListener.

5.2.1 getInetAddress

public InetAddress **getInetAddress** ()

throws `RemoteException`

This method returns the host on which the `Manager` is running.

Returns:

host

Throws:

`RemoteException` - For any communication-related exception.

5.2.2 `getAdapterURLList`

```
public URL[] getAdapterURLList()  
                throws RemoteException
```

This method is used to obtain a set of URLs which can be used to access adapted `MBeans` (i.e. from a web browser).

Returns:

array of URLs

Throws:

`RemoteException` - For any communication-related exception.

5.2.3 `getMBean`

```
public MBean[] getMBean(Class componentType)  
                throws RemoteException
```

This method will return all `MBeans` of the requested type that are registered with the `Manager`.

Returns:

an array of `MBean` instances.

Throws:

`RemoteException` - For any communication-related exception.

5.2.4 `getAllMBeans`

```
public MBean[] getAllMBeans()  
                throws RemoteException
```

This method returns all `MBeans` registered with the `Manager`.

Returns:

an array of `MBean` instances.

Throws:

`RemoteException` - For any communication-related exception.

5.2.5 `addManagerListener`

```
public void addManagerListener(ManagerListener listener)  
                throws RemoteException
```

This method registers an `ManagerListener` which will be notified when MBeans are added or removed from this `Manager`.

Parameters:

`listener` - The listener to be added.

`handback` - An object to be handed back to the listener when the event is reported.

Throws:

`RemoteException` - For any communication-related exception.

5.2.6 `removeManagerListener`

```
public void removeManagerListener(ManagerListener listener)
    throws RemoteException
```

This method unregisters a previously registered `ManagerListener`.

Parameters:

`listener` - The listener to be removed.

Throws:

`RemoteException` - For any communication-related exception.

5.2.7 `getUserInterfaceFactories`

```
public UserInterfaceFactory[] getUserInterfaceFactories(MBean bean)
    throws RemoteException
```

This method is used to obtain user interface factories available that can build a user interface for a particular service interface. After these `UserInterfaceFactory` objects are returned, they can be used to generate the actual user interface.

Parameters:

`bean` - the MBean for which a user interface is needed.

Returns:

array of factories, or `null` if no factories were installed with this `RemoteManager`.

Throws:

`RemoteException` - For any communication-related exception.

See Also:

`UserInterfaceBuilder`

5.2.8 `getUserInterfaceFactories`

```
public UserInterfaceFactory[] getUserInterfaceFactories(
    MBean bean,
    Class[] desiredUserInterfaceClasses)
    throws RemoteException
```

This method is used to obtain user interface factories available that can build a user interface for a particular service interface. The factory must be able to produce a user

interface that is an instance of one of the specified Java types. After these `UserInterfaceFactory` objects are returned, they can be used to generate the actual user interface.

Parameters:

`bean` - the `MBean` for which a user interface is needed.

`desiredUserInterfaceClasses` - array of Java types of user interfaces which are acceptable

Returns:

array of factories, or `null` if no factories were installed with this `RemoteManager`.

See Also:

`UserInterfaceBuilder`

5.3 net.openwings.management.ManagerListener

public interface **ManagerListener**

This interface describes a callback that can be registered with a `Manager` to receive notification when an `MBean` is added or removed from a `Manager` or `RemoteManager`

Method Summary

void	processMBeanAdded (<code>MBean bean</code>) Process an event signalling an <code>MBean</code> was added to the <code>Manager</code> .
void	processMBeanRemoved (<code>MBean bean</code>) Process an event signalling an <code>MBean</code> was removed from the <code>Manager</code> .

5.3.1 processMBeanAdded

public void **processMBeanAdded**(`MBean bean`)
throws `RemoteException`

Process an event signaling an `MBean` was added to the `Manager`.

Parameters:

`bean` - the management bean that was added.

5.3.2 processMBeanRemoved

public void **processMBeanRemoved**(`MBean bean`)
throws `RemoteException`

Process an event signaling an `MBean` was removed from the `Manager`.

Parameters:

`bean` - the management bean that was removed.

5.4 net.openwings.management.Manager

public interface **Manager** extends RemoteManager

This interface is the centerpiece of the Openwings Management architecture. The methods defined in this interface provide access to the management capabilities associated with a component; therefore every Openwings Component will contain a `Manager` object implementing this interface.

A subset of these methods are exposed in the `RemoteManager` interface, a service that allows management to be accessed remotely.

An implementation of this interface is obtained using the `ManagerFactory`. An Openwings Component can obtain a `Manager` object by calling the `net.openwings.component.ComponentComplex.getManager()` method.

Method Summary	
void	addMBean (MBean bean) This method adds an MBean to this Manager.
void	addMBeanAdapter (MBeanAdapter adapter) This method adds an MBeanAdapter to this Manager.
void	addUserInterfaceFactory (UserInterfaceFactory factory) This method adds a user interface factory to this Manager.
void	removeMBean (MBean bean) This method removes an MBean from this Manager.
void	removeMBeanAdapter (MBeanAdapter adapter) This method removes an MBeanAdapter.
void	removeUserInterfaceFactory (UserInterfaceFactory factory) This method removes a user interface factory from this Manager.
void	shutdown () This method is used to terminate all processing provided by a Manager.

5.4.1 addMBean

public void **addMBean** (MBean bean)

This method adds an MBean to this Manager.

Parameters:

bean - The management bean to be added.

5.4.2 removeMBean

public void **removeMBean** (MBean bean)

This method removes an `MBean` from this `Manager`.

Parameters:

`bean` - The management bean to be removed.

5.4.3 addMBeanAdapter

```
public void addMBeanAdapter(MBeanAdapter adapter)
```

This method adds an `MBeanAdapter` to this `Manager`. All existing and `MBeans` in the `Manager` will be passed to the `MBeanAdapter`.

Parameters:

`adapter` - the management bean adapter

5.4.4 removeMBeanAdapter

```
public void removeMBeanAdapter(MBeanAdapter adapter)
```

This method removes an `MBeanAdapter`.

Parameters:

`adapter` - the management bean adapter

5.4.5 addUserInterfaceFactory

```
public void addUserInterfaceFactory(UserInterfaceFactory factory)
```

This method adds a user interface factory to this `Manager`. Factories added to the `Manager` should be used to create user interfaces requested by the inherited method `RemoteManager.getUserInterface()`.

Parameters:

`factory` - the user interface factory

5.4.6 removeUserInterfaceFactory

```
public void removeUserInterfaceFactory(UserInterfaceFactory factory)
```

This method removes a user interface factory from this `Manager`.

Parameters:

`factory` - the user interface factory

5.4.7 shutdown

```
public void shutdown()
```

This method is used to terminate all processing provided by a `Manager`. All registered `MBeans` and `ManagerListeners` are removed, and all `MBeanAdapters` are shutdown. The

RemoteManager service provided by the Manager should be removed. After this method is called, any operation on the Manager is undefined.

5.5 net.openwings.management.ManagerFactory

```
public final class ManagerFactory
```

This class provides a way to obtain an implementation of `Manager`. The choice of `Manager` implementation class is set using the `net.openwings.management.implementation` property.

Method Summary

static <code>Manager</code>	getManager() This method is used to obtain a <code>Manager</code> .
-----------------------------	---

5.5.1 getManager

```
public static Manager getManager()
```

This method is used to obtain a `Manager`.

Returns:

object implementing the `Manager` interface.

5.6 net.openwings.management.MBeanAdapter

public interface **MBeanAdapter** extends `ManagerListener`

This interface defines a plugin for an adapter that makes `MBeans` accessible to another management framework (such as JMX or SNMP). This interface extends `ManagerListener` so that the adapter can be notified of `MBean` additions and removals.

Method Summary	
URL	getAdapterURL() This method is used to obtain a <code>URL</code> which can be used to access the adapted <code>MBeans</code> .
void	shutdown() This method is used to notify an adapter that the <code>Manager</code> is being shut down.

5.6.1 getAdapterURL

public URL **getAdapterURL()**

This method is used to obtain a `URL` which can be used to access the adapted `MBeans`. An example of this would be an `http://` address, which would be used to access the management framework from a web browser.

Returns:

a `URL` that may be used to access the adapted `MBeans`, or `null` if no `URL` is provided.

5.6.2 shutdown

public void **shutdown()**

This method is used to notify an adapter that the `Manager` is being shut down. After this method is called, any operation on the adapter is undefined.

5.7 net.openwings.management.ManagementPolicy

public interface **ManagementPolicy** extends Policy

This interface defines a policy which describes configurable management parameters of a component that should be registered upon initialization of the `Manager`:

- Default `MBeans`
- `MBean` User Interface Factories
- `MBeanAdapters`

Method Summary	
Class []	getMBeanAdapterClasses () This method returns the <code>MBeanAdapters</code> to be used by the <code>Manager</code>
Class []	getMBeanClasses () This method returns the default <code>MBeans</code> to be created by the <code>Manager</code>
UserInterfaceFactory []	getUserInterfaceFactories () This method returns the <code>UserInterfaceFactory</code> s that correspond to the default <code>MBeans</code>
void	setMBeanAdapterClasses (Class [] mBeanAdapterClasses) This method sets the <code>MBeanAdapters</code> to be used by the <code>Manager</code>
void	setMBeanClasses (Class [] mBeanClasses) This method sets the default <code>MBeans</code> to be created by the <code>Manager</code>
void	setUserInterfaceFactories (UserInterfaceFactory [] userInterfaceFactories) This method sets the <code>UserInterfaceFactory</code> s that correspond to the default <code>MBeans</code>

5.7.1 setMBeanClasses

public void **setMBeanClasses** (Class [] mBeanClasses)

This method sets the default `MBeans` to be created by the `Manager`

Parameters:

`mBeanClasses` - array of classes representing the `MBeans` to be created

5.7.2 getMBeanClasses

public Class [] **getMBeanClasses** ()

This method returns the default `MBeans` to be created by the `Manager`

Returns:

array of classes representing the `MBeans` to be created

5.7.3 `setMBeanAdapterClasses`

```
public void setMBeanAdapterClasses(Class[] mBeanAdapterClasses)
```

This method sets the `MBeanAdapters` to be used by the `Manager`

Parameters:

`mBeanAdapterClasses` - array of classes representing the `MBeanAdapters` to be used

5.7.4 `getMBeanAdapterClasses`

```
public Class[] getMBeanAdapterClasses()
```

This method returns the `MBeanAdapters` to be used by the `Manager`

Returns:

array of classes representing the `MBeanAdapters` to be used

5.7.5 `setUserInterfaceFactories`

```
public void setUserInterfaceFactories(UserInterfaceFactory[] userInterfaceFactories)
```

This method sets the `UserInterfaceFactorys` that correspond to the default `MBeans`

Parameters:

`userInterfaceFactoryClasses` - factory objects that may be used to generate user interfaces for `MBeans` in this manager

5.7.6 `getUserInterfaceFactories`

```
public UserInterfaceFactory[] getUserInterfaceFactories()
```

This method returns the `UserInterfaceFactorys` that correspond to the default `MBeans`

Returns:

array of classes representing the `UserInterfaceFactory` objects

6. Tools / Utilities

6.1 Openwings Shell/Explorer

Interaction with Management Beans has been integrated into the Openwings Shell and Openwings Explorer Tools provided as part of the reference implementation. These tools provide hierarchical views of Openwings systems, including contexts, platforms, processes, components, services, and MBeans. MBeans are contained in a directory or sub tree under the components they manage, as shown in the figure below.

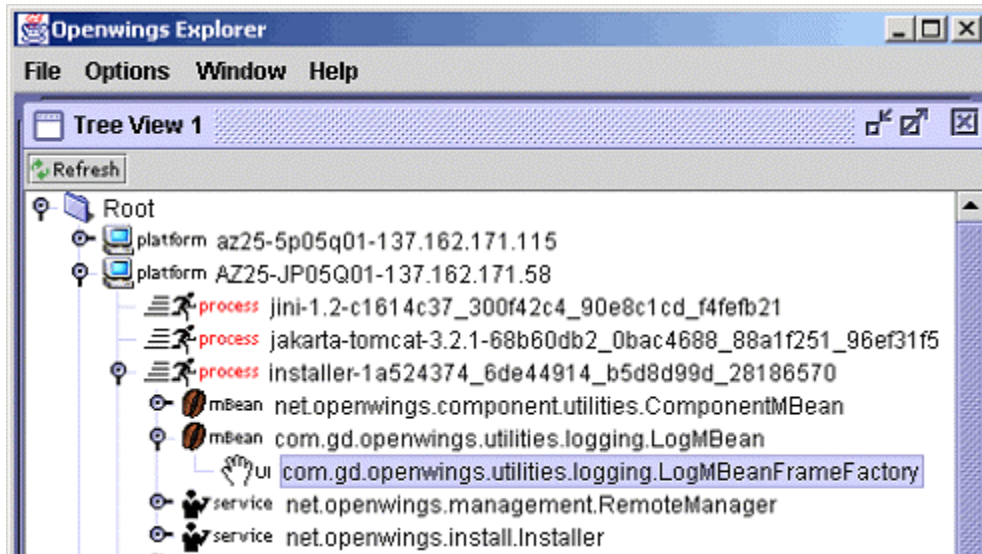


Figure 13: Management Beans in Openwings Explorer

In this figure, the `Installer` process has two MBeans. MBean user interfaces are shown as elements under MBeans. The `LogMBean` for the process has a user interface factory available. Double clicking on the UI element causes the Explorer to display this user interface, as shown in the figure below.

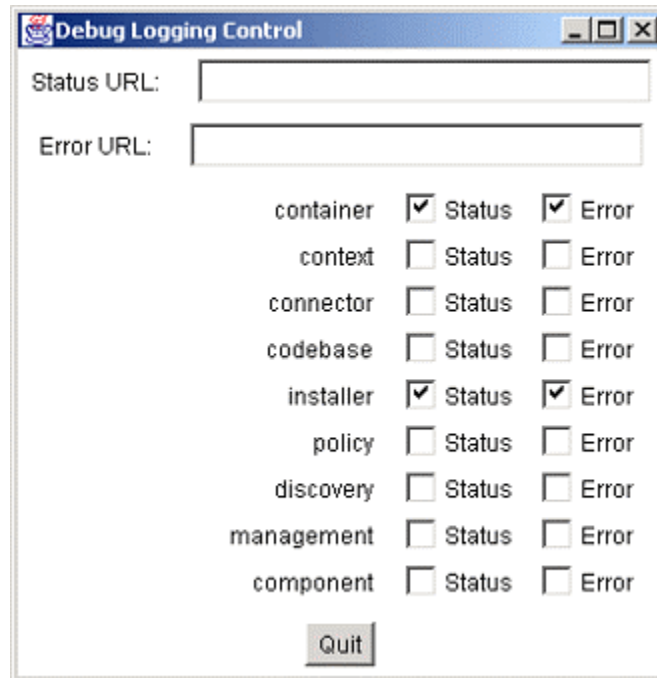


Figure 14: Example MBean User Interface

The management bean and UI in this example control various debug logging levels for the “Installer” process.

7. Examples

7.1 Creating an MBean

To create a management bean for your component, you'll need to start with an interface that exposes some set of manageable aspects of your component.

```
package com.gd.sample;

/**
 * This interface is simply to illustrate a simple management bean.
 */
public interface SampleMBean extends net.openwings.management.MBean
{
    public String getParameter();
    public void setParameter(string value);
}
```

In the same package or another package, also generate an implementation of the MBean interface. Assume that the implementation for our example is called `SampleMBeanImpl`, and that we've created a `net.openwings.ui.UserInterfaceFactory` for our MBean called `SampleMBeanFrameFactory` that builds a `java.swing.JFrame` user interface for a `SampleMBean`. (See the Openwings Component Services specification for more information on how to write a `UserInterfaceFactory`.) Here's some example code that demonstrates how an MBean and the user interface factory are registered.

```
import net.openwings.management.*;

...

Manager manager = ManagerFactory.getManager();
manager.addMBean(new SampleMBeanImpl);
manager.addUserInterfaceFactory(new SampleMBeanFrameFactory);
```

Figure 15: Sample MBean code

In addition to using the programmatic interface, default management beans for a particular component, platform, or even a context may be set using a Management Policy. For example, the Management Policy contained in the Openwings reference implementation contains a management bean called the `com.gd.openwings.utilities.logging.LogMBean` to control the debug logging used throughout the reference implementation.

7.2 Creating an MBeanAdapter

In order to integrate Openwings MBeans with other management frameworks, you may decide to create an `MBeanAdapter` to represent MBeans to another framework. To create an adapter, implement the MBean adapter interface.

Assume that the adapter implementation for our example is called `SampleMBeanAdapter`. Here's some example code that demonstrates how to register the adapter with Management Services.

```
import net.openwings.management.*;
...
Manager manager = ManagerFactory.getManager();
manager.addMBeanAdapter(new SampleMBeanAdapter);
```

Figure 16: Sample MBeanAdapter Code

In addition to using the programmatic interface, default management beans for a particular component may be set using a `ManagementPolicy`.

8. Compliance

The following rules are required for compliance to this specification:

#	ITEM
1	Comply with the interfaces defined in this document.
2	Pass the validation suite for this service.

Figure 17: Compliance Table

9. References and Further Reading

1. Openwings Architecture Whitepaper,
<http://www.openwings.org/download/specs/openwingswp.pdf>
2. Openwings Architecture Specification,
http://www.openwings.org/download/specs/Openwings_Architecture_Description.pdf
3. Openwings Component Services Specification,
http://www.openwings.org/download/specs/Openwings_Component_Services.pdf
4. Openwings Policy Services Specification,
http://www.openwings.org/download/specs/Openwings_Policy_Services.pdf
5. Openwings Interface Definition Specification,
http://www.openwings.org/download/specs/openwings_interface.pdf
6. Web Based Enterprise Management (WBEM), <http://www.dtmf.org/>
7. Java Dynamic Management Kit (JDMK), <http://www.sun.com/software/java-dynamic/wp-jdmk.kit>
8. Java Management Extensions (JMX),
<http://java.sun.com/products/JavaManagement/index.html>
9. JIRO, <http://www.jiro.org>