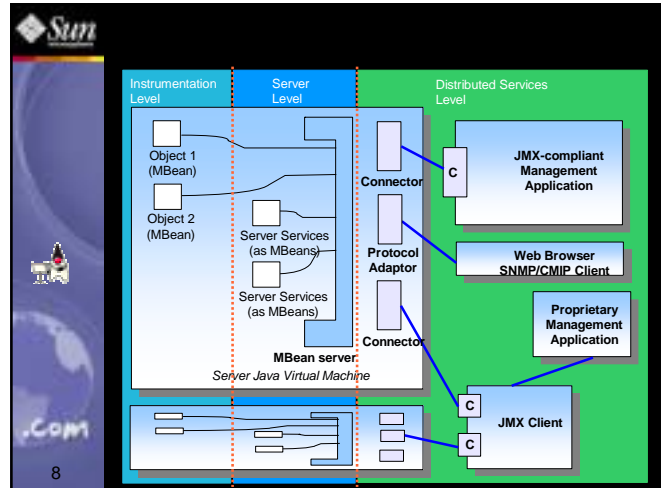

 The JMX architecture defines three levels:

- Instrumentation Level
  - How to instrument managed resources
- Server Level
  - How managed resources are managed via the management server
- Distributed Services Level
  - How distributed clients and management applications access and interact with servers and the managed resources in the servers


7



 **Introduction**

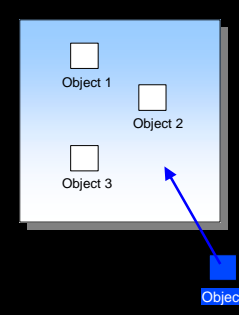
- Architecture
- Instrumentation Specification
- Server Specification

9


 **Basic Goal**

*In a running Java application, we would like to be able to:*

- Manage existing Java objects:
  - get an attribute value
  - change an attribute value
  - invoke an operation
- Add new Java objects:
  - using existing Java classes
  - using new classes from an arbitrary location
- And do all this from a remote location

 The diagram shows a Java application window containing three objects: Object 1, Object 2, and Object 3. A blue arrow points from a box labeled Object 4 (representing a remote location) to Object 2, indicating remote access.

10


 **Client:** Application controlling servers by submitting requests, activating new services, etc ...

**Server:** Application making local resources available remotely and providing one or more services

**MBean server:** Registry for MBeans in the server

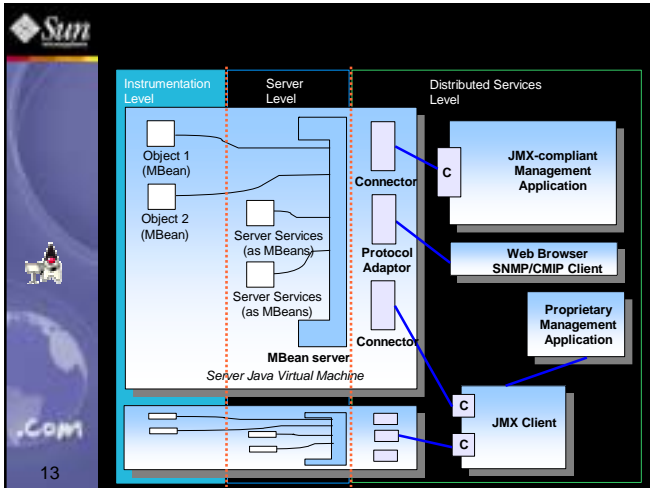
**MBean:** Managed bean. Java class implementing a management interface and representing a resource to be managed or monitored

11

 **Introduction**

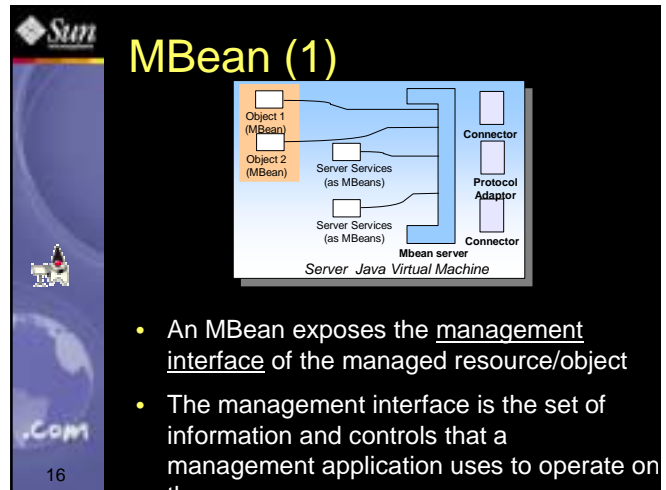
- Architecture
  - Instrumentation Level
  - Server Level
  - Distributed Services Level
- Instrumentation Specification
- Server Specification

12



- Provides management agent independence
- Specification for implementing JMX manageable resources
- Resource can be a service, device, user, ... etc.
- Resource developed in Java or have Java wrapper


- MBeans
  - Defines how resources are instrumented using MBeans
  - MBean instrumentation allows the resource to be manageable through JMX-compliant agents
- Notification Model
  - MBeans and other JMX components may emit notifications
- MBean Metadata Classes
  - Describes the MBean's management interface for management agents and management applications





- An MBean exposes the management interface of the managed resource/object
- The management interface is the set of information and controls that a management application uses to operate on the resource

- An MBean exposes the management interface as:
  - Attributes which may be accessed
  - Operations which may be invoked
  - Notifications which may be emitted (optional)
  - Constructors for the MBean's Java class
- An MBean follows design patterns
  - The way an attribute or operation to be exposed has to be declared
- An MBean must be registered in the MBean server to be visible remotely


- Standard MBeans
  - Simplest to design and implement
  - Management interface described by method names, like getState, setState, reset
- Dynamic MBeans
  - Must implement DynamicMBean interface
  - Expose management interface at runtime for greater flexibility via generic methods, like getAttribute, setAttribute, invoke
- Model MBeans are dynamic MBeans
  - Self contained, self described, configurable at runtime
  - Generic MBean class for dynamic instrumentation of resources
- Open MBeans are dynamic MBeans
  - Basic data types only for universal manageability





- JMX defines a generic notification model
  - Based on Java event model
- Notification can be emitted by:
  - MBean instances
  - MBean server
- JMX defines
  - Notification objects
  - Broadcaster interface for notification senders
  - Listener interface for notification listeners
- Distribution of notification for remote management applications not in JMX 1.0


19





- MBean server (server level) has to provide MBean metadata
- Classes that describe the management interface of an MBean
- There are metadata classes for attributes, operations, notifications, and constructors
- Metadata include name, description, attribute type, operation parameter types, ... etc.
- Different types of MBeans extend metadata classes to provide additional information

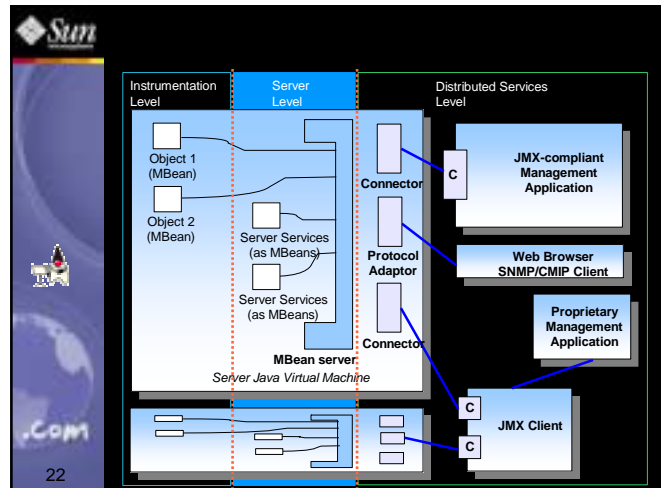
20




- **Introduction**
- Architecture
  - Instrumentation Level
  - Server Level
  - Distributed Services Level
- Instrumentation Specification
- Server Specification



 

21






- Specification for implementing servers
- Servers directly control resources and make them available to remote management applications
- JMX server consists of:
  - MBean server
  - Server services
- Implemented by developers of management systems
  - Independent of semantics of JMX manageable resources or management application



 

23




## MBean Server (1)


- The MBean server is a registry of objects which are exposed to management operations in a server
  - Only registered MBeans can be managed from outside the server's JVM
- The MBean server only exposes an MBean's management interface (not the MBean's direct reference)


24

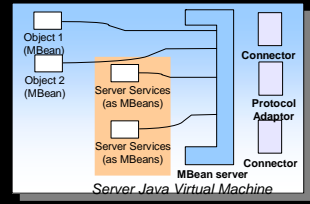


- Also provides standardized interface for accessing MBeans within the same JVM
- MBeans can be instantiated and registered by:
  - Another MBean
  - The management server itself
  - A remote management application (through distributed services)
- Each registered MBean is assigned a unique name, usually user-defined
- Only these names are used by server/client applications to identify the MBeans
- Relies on protocol adaptors and connectors to make server accessible from outside server's JVM


 .com

25







- Objects that perform management operations on the MBeans registered in the MBean server
  - Provides management intelligence inside the server
  - Server services are also MBeans

 .com

26



- M-Let dynamic class loading service retrieves and instantiates new classes from arbitrary network locations
- Monitors observe MBean attribute value and emit notifications for several types of changes in the target
- Timers provide scheduling based on a one-time alarm clock or repeated, periodic notification
- Relation service defines associations between MBeans and enforces the cardinality of the relation based on

 .com


27

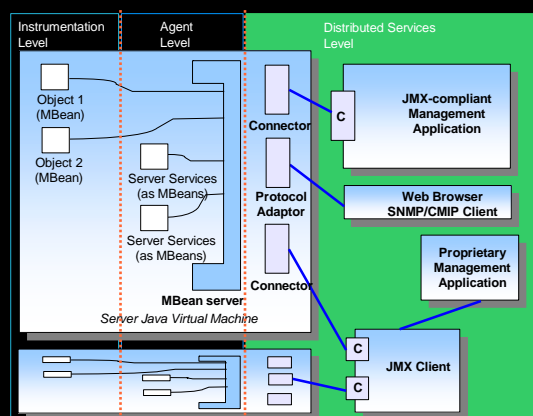



- **Introduction**
- Architecture
  - Instrumentation Level
  - Server Level
  - **Distributed Services Level**
- Instrumentation Specification
- Server Specification

 .com


28






 .com

29



- Outside scope of JMX 1.0
- Defines distribution mechanism
  - Connectors
  - Protocol Adaptors
  - Proxies
- Specifies interfaces for JMX clients
- Defines management interfaces and components that operate on servers or hierarchies of servers

 .com

30

## Connector (1)

- Composed of a client part and a server part, the connector connects a JMX client application with an MBean server
- Client part provides a protocol-independent API to access the remote MBean server
- Tunnels management operations over a specific protocol (RMI, HTTP, ...)

31

## Connector (2)

- Provides remote access to MBean attributes and operations
- Provides remote instantiation, registration and de-registration of MBeans
- Propagates MBean and MBean server notifications to the client application

32

## Protocol Adaptor

- Only has a server part, on the server
- Creates a view of the MBeans through a given protocol and information model

33

## Connectors and Protocol Adaptors

Not Shown: HTTPS, IIOP (JMX IIOP JSR)

Client API

- RMI: Java Application
- HTTP: Java Application
- HTML: HTML Browser: HotJava, Netscape, ...
- SNMP: SNMP Client: SNM, SEM, JDMK, ...

34

- The proxy MBean provides an image of an MBean as a stub object on the client side
- Operations performed on the proxy MBean are propagated to the MBean
- Notifications emitted by the MBean are propagated to the proxy MBean
- The proxy is generated from an MBean


35

## Development Process

### Main Steps

- Instrument managed resources
- Instantiate MBeans in a JMX server
- Test MBeans
- Develop the client application (optional)

36




Make an existing Java application manageable

- The existing application can be modified so that
  - It creates an MBean server and a connector or a protocol adaptor
  - It registers the objects that need to be managed
- Requirement: registered objects must be instrumented as MBeans
  - This requirement is light

It does not break existing design or force the


37



Open the design of new Java applications


- Design the new application such that it includes an MBean server
  - Remote access for management operations is free
  - The application will be extensible dynamically
- Develop servers for a variety of purposes
  - System Management/Monitoring
  - Network Management/Monitoring
  - Application Management/Monitoring

38




- Protocol independent
- Information model independent
- Light requirements on software design:
  - An MBean only needs to implement a management interface
  - An MBean does not need to be aware of JMX servers and agents
  - An MBean does not need to know anything about communication with the outside world
- Scalability and flexibility
  - By registering and unregistering MBeans the Java application can include only what it needs at the time it needs it
  - New protocol adaptors (with new protocols) can

39




- **Introduction**
- Architecture
- **Instrumentation Specification**
  - Standard MBean
  - Dynamic MBean
  - MBean Metadata
  - Notification Model
  - Model MBean
- Server Specification

40




- Defined by the JMX Instrumentation Spec
- Must follow specified
  - Design patterns
  - Implement MBean interfaces
- MBean enable management by any JMX servers
- An MBean can be created from an already existing Java object or from scratch
- Developers decide
  - Granularity of MBeans

41




- MBean must be a public concrete Java class
- Must implement one of two types of MBean interfaces
  - **Standard MBean interface**
    - Management interface is **static**, i.e. determined from methods defined in the MBean's interface
    - Depends on design patterns and introspection
  - DynamicMBean interface
    - Management interface is **generic** and determined indirectly through method calls
    - Generic get, set, and invoke methods
- Optional NotificationBroadcaster interface

42

 .Com


- An attribute is a field or property of a JMX manageable resource exposed via an MBean's management interface
- An operation is an action that a JMX manageable resource makes available to management applications exposed via an MBean's management interface
- A method refers to a Java method or function

43

 .Com


- Choice of standard MBean or dynamic MBean interface determines how MBean will be developed, not how it will be managed
- JMX server provides abstraction to manage both types of instrumentation transparently
- Management applications handle them in a similar manner

44

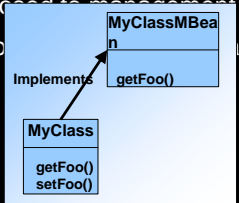
 .Com

- **Introduction**
- Architecture
- Instrumentation Specification
  - **Standard MBean**
  - Dynamic MBean
  - MBean Metadata
  - Notification Model
  - Model MBean
- Server Specification


45

 .Com

- Implement Java interface named after the class
  - Name of MBean interface is formed by adding MBean suffix to class name
- Standard MBean interface lists the attributes and operations exposed to management
- Impose very little code on class design



46

 **Lexical design patterns for methods**


- An attribute is deduced from the existence of a **get** and/or **set** method:

```
public MyType getMyAttribute();
public void setMyAttribute(MyType value);
```

- Any other method is considered to be an operation:

```
public void reset();
public MyResult myOperation(MyType arg);
```

47


 .Com

```
public interface SimpleStandardMBean {
    public String getState();
    public void setState(String s);
    public Integer getNbChanges();
    public void reset();
}

public class SimpleStandard implements SimpleStandardMBean {
    public String getState() {
        return state;
    }
    public void setState(String s) {
        state = s;
        nbChanges++;
    }
    public Integer getNbChanges() {
        return new Integer(nbChanges);
    }
    public void reset() {
        state = "initial state";
        nbChanges = 0;
        nbResets++;
    }
    public Integer getNbResets() {
        return new Integer(nbResets);
    }
    private String state = "initial state";
    private int nbChanges = 0;
    private int nbResets = 0;
}
```


- **Attribute** `state`  
Type `String`  
Access `read-write`
- **Attribute** `NbChanges`  
Type `Integer`  
Access `read-only`
- **reset** is an manageable operation
- **NbResets** is not a manageable attribute

48



- Introduction
- Architecture
- Instrumentation Specification
  - Standard MBean
  - Dynamic MBean
  - MBean Metadata
  - Notification Model
  - Model MBean
- Server Specification

49




### Standard MBean

- Suitable when structure of management data is well-defined in advance
- Determine structure at compile time
- Provide quick and easy way to instrument manageable resources

### Dynamic MBean

- Suitable when structure of management data is likely to evolve often over time
- Can determine structure dynamically at run-time
- Provide adaptability and more elaborate management capabilities

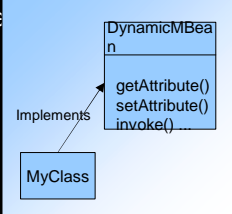
50




## Dynamic MBeans

### Management Interface

- Implement DynamicMBean interface
- Provide generic get, set, invoke methods
- Describe their management interface at runtime



51




«Interface»  
**DynamicMBean**

```

getMBeanInfo(): MBeanInfo
getAttribute( attribute:String)Object
getAttributes( attributes:String[])AttributeList
setAttribute( attribute:Attribute, void)
setAttributes( attributes:AttributeList)AttributeList
invoke( actionName:String,
      params:Object[],
      signature:String[] ) Object

```

52



Definition of a dynamic MBean with:


- one read-write attribute State
- one read-only attribute NbChanges
- one reset operation

```

public class SimpleDynamic implements DynamicMBean {
    public SimpleDynamic() {
        // Creates MBeanAttributeInfo for "State" and "NbChanges"
        // Creates MBeanOperationInfo for "reset"
    }
    public MBeanInfo getMBeanInfo() {
        // Creates a MBeanInfo with the name "SimpleDynamic" and the
        // MBeanAttributeInfo and MBeanOperationInfo
    }
    public Object getAttribute(String attribute) {
        // Checks if the requested attribute is "State" or "NbChanges"
        // Returns the attribute value.
    }
    public void setAttribute(Attribute attribute) {
        // Checks if attribute is "State" and if argument
        // is a String. Sets the value of the attribute.
    }
}

```

53



```

public AttributeList getAttributes(String[] attributes) {
    // Checks if "State" and/or "NbChanges" is in the list.
    // Calls getAttribute.
}
public void setAttributes(AttributeList attribute) {
    // Checks if "State" is in the list.
    // Calls setAttribute.
}
public Object invoke(String opName, Object[] args,
                    String[] signature) {
    // Checks the operation is "reset",
    // with the correct signature (no args).
    // Perform reset
}
...
}

```

54

```

Sun
public Object getAttribute(String attributeName)
throws AttributeNotFoundException,
MBeanException,
ReflectionException {
    // Check attributeName is not null to avoid
    // NullPointerException later on
    if (attributeName == null) {
        throw new RuntimeOperationsException(
            new IllegalArgumentException(...), ...);
    }
    // Check for a recognized attribute_name and
    // call the corresponding getter
    if (attributeName.equals("State")) {
        return getState();
    }
    if (attributeName.equals("NbChanges")) {
        return getNbChanges();
    }
    // If attributeName has not been recognized throw
    // an AttributeNotFoundException
    throw new AttributeNotFoundException(...);
}
.com
55

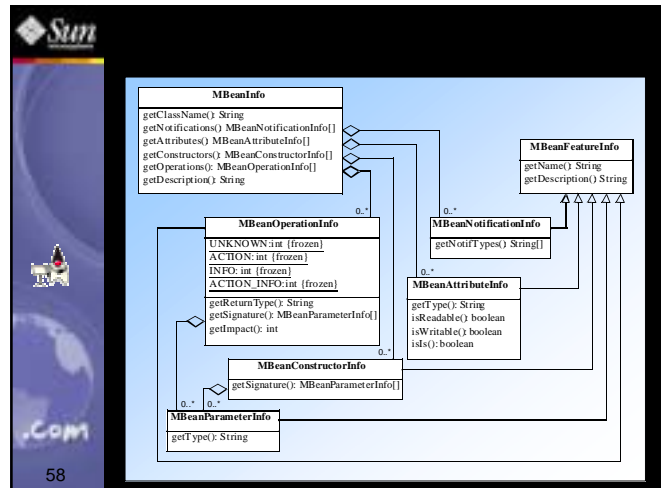
```

```

Sun
public Object invoke(String operationName,
Object params[],
String signature[])
throws MBeanException,
ReflectionException {
    // Check operationName is not null to avoid
    // NullPointerException later on
    if (operationName == null) {
        throw new RuntimeOperationsException(
            new IllegalArgumentException(...), ...);
    }
    // Check for a recognized operation name and call the
    // corresponding operation
    if (operationName.equals("reset")) {
        reset();
        return null;
    } else {
        // unrecognized operation name:
        throw new ReflectionException(
            new NoSuchMethodException(operationName), ...);
    }
}
.com
56

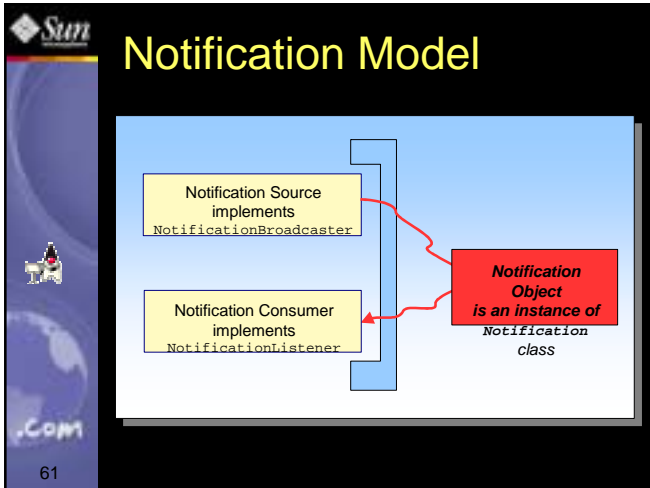
```

- Introduction
  - Architecture
  - Instrumentation Specification
    - Standard MBean
    - Dynamic MBean
    - **MBean Metadata**
    - Inheritance Pattern
    - Notification Model
    - Model MBean
  - Server Specification
- .com
57



- Coherence
    - Developer must ensure that advertised management interface (via getMBeanInfo) is accurately mapped to internal implementation
    - MBean server does not test or validate the self-description of a dynamic MBean
  - Dynamics
    - The management interface of an MBean should be applicable throughout the lifetime of the MBean
    - getMBeanInfo should return the same MBeanInfo during the lifetime of the dynamic MBean
- .com
59

- Introduction
  - Architecture
  - Instrumentation Specification
    - Standard MBean
    - Dynamic MBean
    - MBean Metadata
    - Inheritance Pattern
    - **Notification Model**
    - Model MBean
  - Server Specification
- .com
60



- Notifications are instances of `Notification`
    - Can be used directly or may be subclassed
  - A `Notification` object contains the following fields:
    - An event source references the notification source (from `java.util.EventObject` superclass)
    - A notification type is a string used to characterize a generic notification object
    - A sequence number is a serial number identifying a notification instance within the context of a notification broadcaster
    - A time stamp indicates when the notification was generated
    - A message containing a string which could be an explanation of the notification for display to the user
- 62

- Must be implemented by all MBeans expecting to receive notifications
- ```
void handleNotification(
    Notification notification,
    Object handback)
```
- Invoked by notification broadcaster MBean when it emits a notification
  - A single notification handler receives all notifications from all potential broadcasters
    - All notifications are characterized by notification type string
  - Handback object
    - The object that the listener provides when the listener registers for notifications with the notification broadcaster
    - Stored by notification broadcaster and returned unchanged with each notification
- 63

- A notification filter object is provided by the listener when the listener registers for notifications with the broadcaster
  - `boolean isNotificationEnabled(Notification n)`  
Invoked by notification broadcaster before it emits a notification
  - Listeners rely on a filter to screen notifications
  - An object can be both a `NotificationListener` and `NotificationFilter`
- 64

- `MBeanNotificationInfo[] getNotificationInfo()`  
**Provides the description** of all notifications this broadcaster emits
  - `void addNotificationListener(NotificationListener listener, NotificationFilter filter, Object handback)`  
Used by listener to register for notifications  
(listener, filter, handback) tuple stored  
Key is (listener, handback)
  - `void removeNotificationListener(NotificationListener listener, Object handback)`  
Remove (listener, filter, handback) tuple
- 65

```

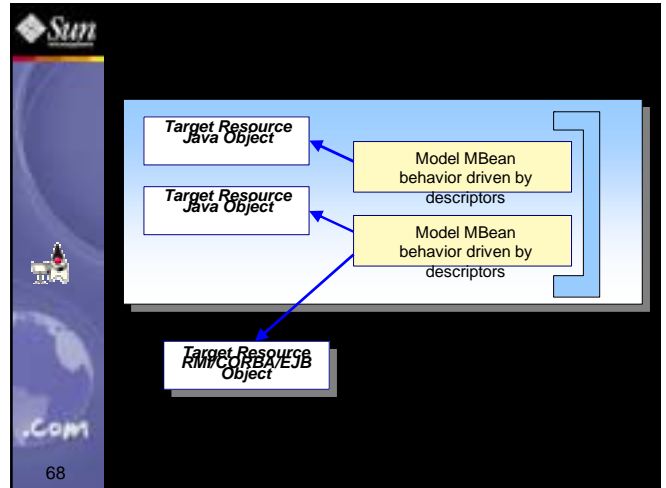
For each notification
  For each (listener, filter, handback)
    If filter.isNotificationEnabled(notification) then
      listener.handleNotification(notification, handback)
    EndIf
  EndFor
EndFor
  
```

66

**Sun**

- Introduction
- Architecture
- Instrumentation Specification
  - Standard MBean
  - Dynamic MBean
  - MBean Metadata
  - Inheritance Pattern
  - Notification Model
  - **Model MBean**
- Server Specification

67



**Sun**

A model MBean is a generic, configurable dynamic MBean

- Concrete model MBean classes provided in conjunction with the JMX server
  - JMX server must provide `RequiredModelMBean`
- Configured via descriptors
- Descriptors contains dynamic, extensible, and configuration behavior information which includes:
  - Persistence policies, value caching policies, logging policies
  - Mapping from MBean attributes and operations to methods on underlying Java object(s)

```

getattribute("myApplStatus")
==> myAppl.statusChecker()
  
```

69

**Sun**

- Resource obtains access to MBean server
- Resource invokes MBean server to create new or find existing model MBeans
- Resource configures model MBeans by manipulating the model MBean descriptors
- Get attribute invoked on a model MBean
  - Model MBean has caching mechanism for attributes
  - Return cached values or delegate invocation to resource object
- Operation invoked on a model MBean
  - delegate to managed resource object
- Managed resource sends notifications by invoking `sendNotification` method of model MBean

70

**Sun**

- Introduction
- Architecture
- Instrumentation Specification
- **Server Specification**
  - MBean Server
  - M-Let Service
  - Monitor Service
  - Timer Service
  - Relation Service

71

**Sun**

## Server Specification

The diagram shows the architecture of the MBean server. On the left, 'Object 1 (MBean)' and 'Object 2 (MBean)' are connected to 'Server Services (as MBeans)'. These services are connected to the 'MBean server'. The 'MBean server' is connected to 'Connectors', which are in turn connected to 'Protocol Adaptors'. The entire system is labeled 'Server Java Virtual Machine'.


**MBean server**

- MBean registry
- All management operations go through MBean server

**Server services**


- Dynamic class loading
- Monitoring
- Timer
- Relation

72




- Consists of two parts
  - Domain name
    - Provides for name spaces within a JMX server or for a more global management solution
    - MBean server can provide default domain
  - Key Property List
    - Unordered list of property-value pairs
    - Used to assign unique names within a domain
    - Must contain at least one key property
- String representation
  - `[domainName]:property=value[,property=value]*`
  - Canonical name: key properties sorted in lexical

73




- Used in queries to identify target objects to limit scope of query
- Domain name
  - \* matches any character sequence, including empty
  - ? matches any one single character
- Key property list
  - \* wildcard for key properties, replaces any number of key properties
- Examples
  - MyDomain:description=printer,type=ink
  - \*:\* all objects in MBean server
  - :\* all objects of the default domain
  - ??Domain:\* all objects in MyDomain
  - \*Domain:\* all objects in domain and those with Domain

74




- ObjectInstance class
  - Contains
    - Class name
    - ObjectName
  - Equality test with another ObjectInstance
  - Returned when an MBean is created and is used subsequently for querying
- Attribute class
  - represents single attribute-value pair
- AttributeList class
  - represents a list of attribute-value pair

75




- Introduction
- Architecture
- Instrumentation Specification
- Server Specification
  - MBean Server
  - M-Let Service
  - Monitor Service
  - Timer Service
  - Relation Service

76




- Only has static methods to
  - Create new MBean server instances
    - MBean server referenced by domain name
  - Find MBean server instances
  - Release MBean server reference

77




- Object creation methods:
  - `createMBean`  
instantiates and registers an MBean
  - `instantiate`  
instantiates an MBean but does not register MBean
  - `deserialize`  
byte array in context of MBean class loader
- MBean registration methods:
  - `registerMBean`
  - `unregisterMBean`
  - `isRegistered`

78



- MBean proxy methods:
  - getMBeanInfo
  - getAttribute      getAttributes
  - setAttribute      setAttributes
  - invoke
  - addNotificationListener
  - removeNotificationListener
- Query method:
  - queryMBeans(ObjectName n, QueryExp q)  
returns Set of ObjectInstance's
  - queryNames(ObjectName n, Query q)  
return Set of ObjectName's
- Read-only attributes:

79



Takes two arguments: ObjectName and QueryExp


- Object name provides scope of query
  - Object name pattern matching selects MBeans on which query expression will be applied
- Query expression operates on MBean attributes
  - Applies to a single MBean
  - Use Query class to build QueryExp

```

QueryExp exp =
    Query.and(
        Query.between(Query.attr("age"),
            Query.value(20),
            Query.value(30)),
        Query.match(Query.attr("name"),
            Query.value("G*ling")))

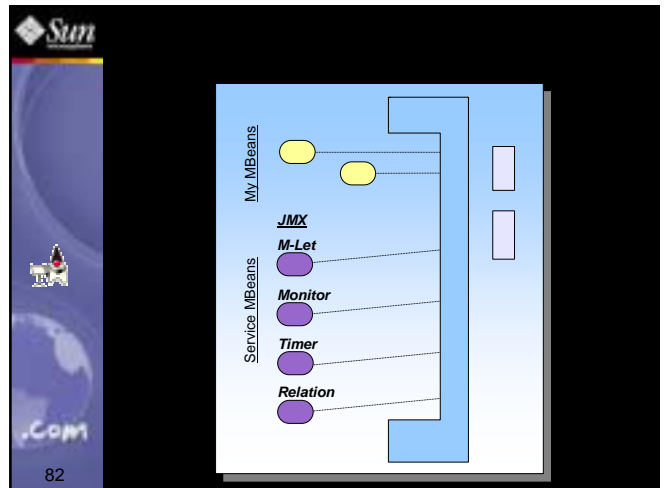
```


80



- Introduction**
- Architecture
- Instrumentation Specification
- Agent Specification
  - MBean Server
  - M-Let Service**
  - Monitor Service
  - Timer Service
  - Relation Service
- Conclusion

81





- MBeans can be obtained from a remote JAR file


```

http://soft.eng/mybean.html
<MLET
CODE=object4.class
ARCHIVE=mybean.jar
NAME="test:name=object4">
</MLET>

http://soft.eng/mybean.jar
object3.class
object4.class

```

83



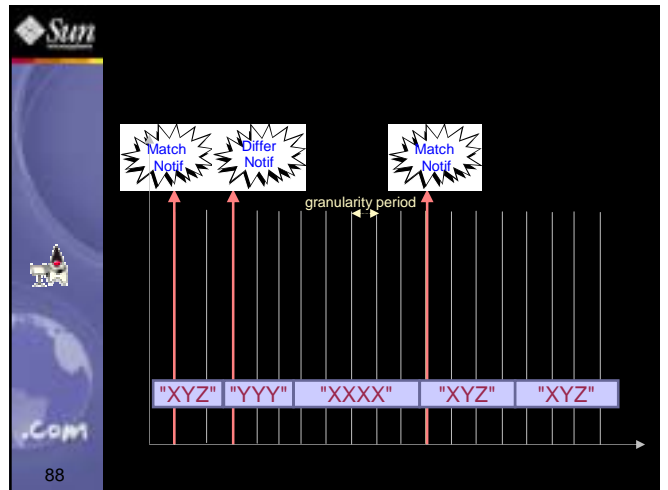
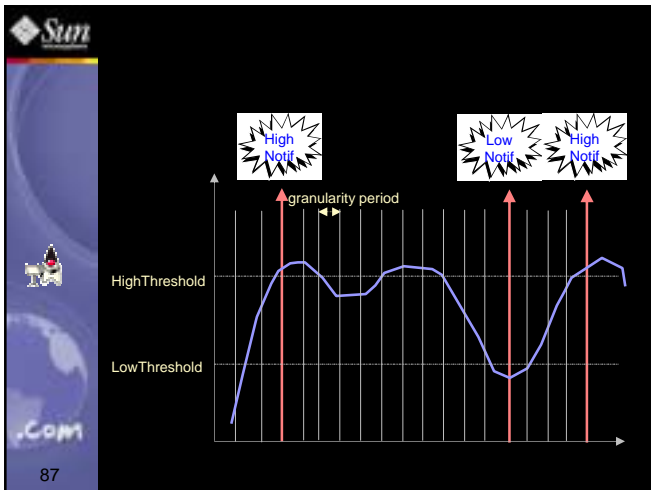
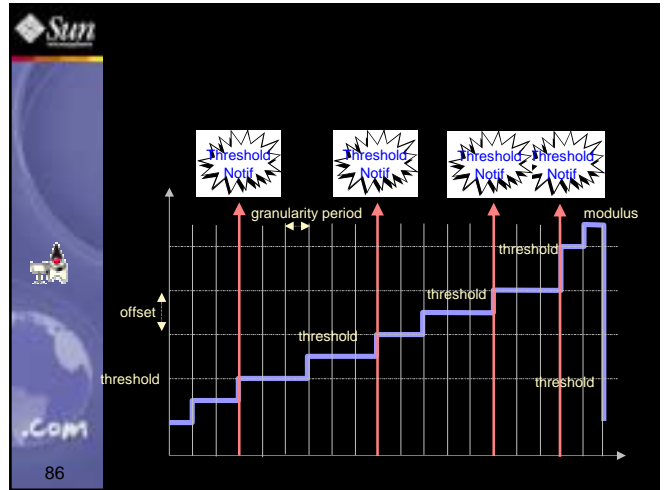
- Introduction**
- Architecture
- Instrumentation Specification
- Server Specification
  - MBean Server
  - M-Let Service
  - Monitor Service**
  - Timer Service
  - Relation Service

84

**Sun**

- A monitor MBean observes an attribute of another MBean at regular interval specified by the granularity period
- The derived gauge is either
  - the exact value, or
  - the difference between two consecutive values
- A MonitorNotification is sent when the attribute value crosses a level or threshold, or string value changes
- There are several types of monitors:
  - Counter Monitor
  - Gauge Monitor

85



**Sun**


- **Introduction**
- Architecture
- Instrumentation Specification
- Server Specification
  - MBean Server
  - M-Let Service
  - Monitor Service
  - **Timer Service**
  - Relation Service

89


**Sun**


- Enables notifications to be sent to all registered listeners
  - At a specified date
  - Starting at a specified date and repeating several times, at regular time intervals
- addNotification
- removeNotification
- SendPastNotification attribute
  - If true, send past notifications when timer restarted and update to the next notification date
  - if false, update past notification dates to next notification date but don't send past

90


 Sun

- **Introduction**
- Architecture
- Instrumentation Specification
- Server Specification
  - MBean Server
  - M-Let Service
  - Monitor Service
  - Timer Service
  - **Relation Service**


 TM


 Sun

91


 Sun

- Used to define logical relations between MBeans
- Act as a repository for relation types and relations
- To create relation between MBeans:
  - Create a new relation type, or use existing one
  - Create a relation from this relation Type
  - ---> will associate MBeans to this relation
- Provides a set of operations to get information about relations and related MBeans

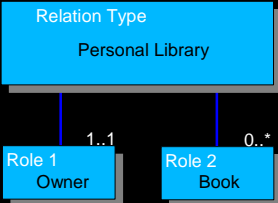
 TM

 Sun

92


 Sun


- Create relations between the Owner and its Books in a Personal Library




```

classDiagram
    class PersonalLibrary["Relation Type Personal Library"]
    class Owner["Role 1 Owner"]
    class Book["Role 2 Book"]
    PersonalLibrary "1..1" -- "0..*" Owner
    PersonalLibrary "1..1" -- "0..*" Book
  
```


 TM


 Sun

93


 Sun

- Only registered MBeans can participate in relations
  - Relation service operates only on object names
  - Provides object names in response to queries
- Consistency checks when
  - Relation type is created
    - e.g. role information inconsistent
  - Relation is created
    - e.g. does not confirm to relation type or object name does not exist
  - Role values (MBeans in role) are updated
- Removal semantics
  - Remove relation type removes all relation of relation type
  - Remove relation does not de-register MBeans
  - De-register an MBean removes the MBean from all role values
  - PurgeFlag attribute and purgeRelations method


 TM


 Sun

94

 Sun

- Common use is to maintain consistency when an MBean is de-registered
  - Trigger code execution when a relation has been updated or deleted due to an MBean de-registration
  - For example, to trigger code execution when resources are unplugged or removed
- Only guarantees that an MBean satisfies its role
  - Cannot define how many relations an MBean may appear in

 TM

 Sun

95

 Sun

**JMX**

- **Home page (Specification, RI, and related JSR's)**  
<http://java.sun.com/products/JavaManagement>
- **Email archives**  
<http://archives.java.sun.com/jmx-forum.html>

**Sun JDMK**

- **Home page**  
<http://www.sun.com/software/java-dynamic>
- **Email archives**  
<http://archives.java.sun.com/jdmk-forum.html>

**IBM Tivoli JMX**

- <http://www.alphaworks.ibm.com/tech/TMX4J>

**AdventNet Agent Toolkit**

- <http://www.adventnet.com/products/agenttoolkit>

 TM

 Sun

96

