Manageability Services Broker

The Open Group Manageability Working Group

Diverse Applications

- Components are 'diverse' across
  - language
  - operating system
  - physical system
  - middlewares
  - networks
  - component protocols
  - corporations
- Applications are critical to business
- Requires manageability

Management today

- Creating Application specific management system
- Creating their own agent type infrastructure
  - Have to learn management principles
  - Only do whats absolutly necessary
  - Often inferior design and capability
  - Can't be accessed by 3rd party management systems
- They would rather not write their own, they would rather someone give them something standard and free.

As management vendors...

- We want to give them a manageability infrastructure suitable for them to use for their own specific management system.
- We can access this infrastructure in terms we understand.
- We can be sure the infrastructure is reliable with reasonable quality.
- We can guide the application developers on managability development
  - Consider elements of manageability (deploy, install, cfg, metrics, ops, events,...)
  - How to expose these elements

The futures so bright...

- As the standard manageability infrastructure becomes pervasive, a vast 'distributed data base' of management information accumulates and makes more advanced and proactive management applications possible
- Sets the stage for much more interesting management solutions
  - dynamic application networks
  - intelligent application networks
  - correlation
  - root cause analysis
  - automated recovery of failures, etc.

Management Services Broker

- Instrumentation use directly by managed resources
- Adaption from instrumentation APIs
- Connection into management systems
- Plug&Swap manageability services
  - substitute required services
  - add support defined standard interface
  - add custom services/custom interfaces
- Define minimum required services
- Define common optional services
- Based on DMTFs WBEM work

Success Factors

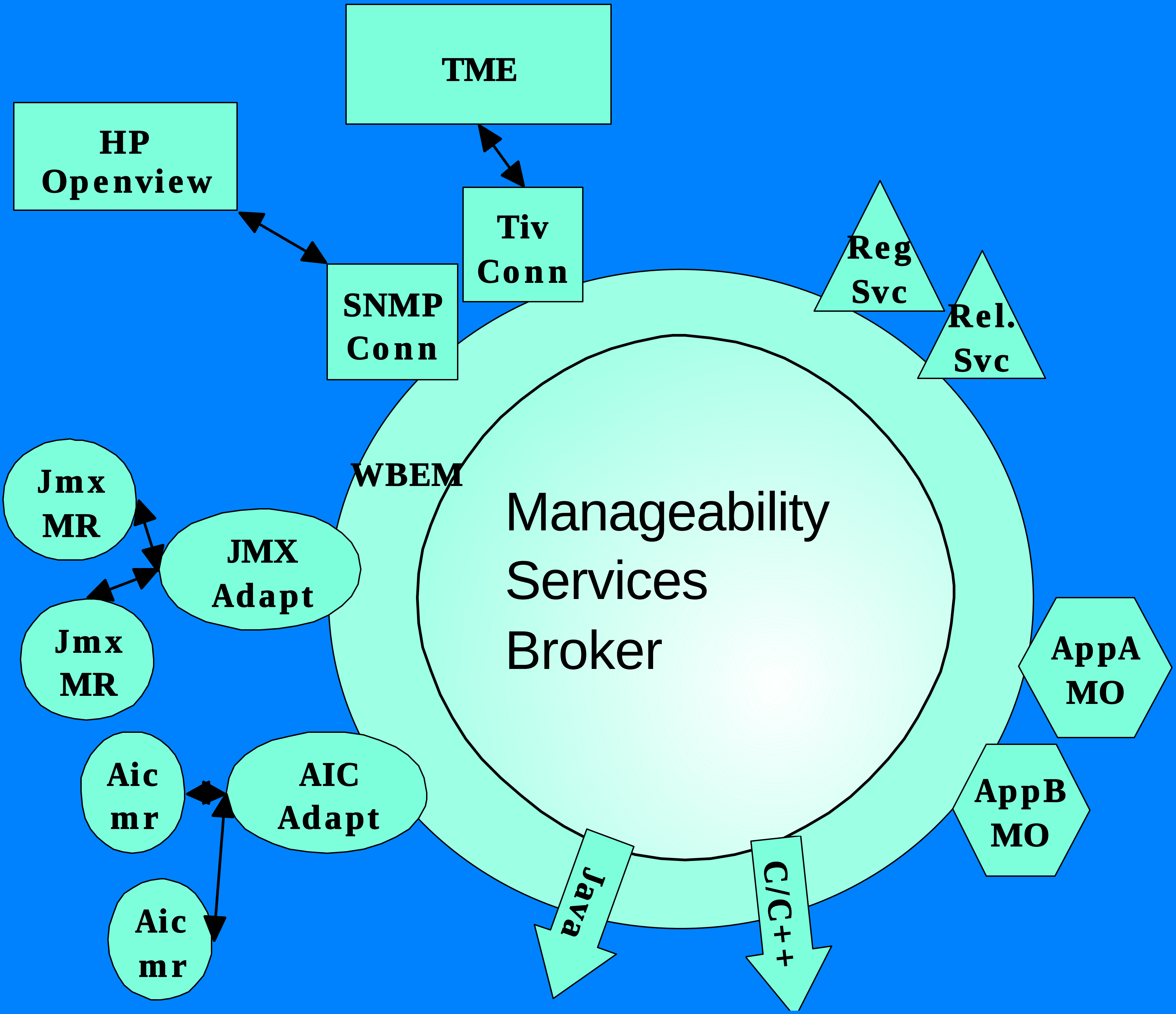
- Easy for application developer
- Flexible and able to support complex applications
- Investment protection for existing instrumentation development (AIC, JMX)
- Support emerging development models (over traditional ones)
- Involvement and support from industry

Standards Involvement

- The Open Group Enterprise Management Program - Manageability Work Group
  - Publisher of Spec
  - Publisher of Open Source Implementations
- DMTF - CIM/WBEM
  - Use xmlCIM, HTTP/Ops, Schema
  - Model for runtime application management
- OMG- Management SIG - interested in CORBA based application management.
  - Same goals, can we cooperate?
  - Brings middleware vendors, distrib app vendors
- OASIS - ebXML is being developed as a protocol and basis for B2B applications.
  - We need to ensure that manageability of these applications is being considered.

Corporation Involvement

- Management Vendors: Tivoli, CA, BMC, HP, Hitatchi(got these thru TOG/DMTF)
- Middleware Vendors: IBM/WebSphere, BEA, Oracle, Inprise, Iona
- Development Tool Vendors: IBM/VisualAge, JBroker, JBuilder, Symantec, etc.
- Application Vendors:
  - Traditional: PeopleSoft, SAP
  - B2B: I2, Ariba, CommerceOne
  - Corporate: Boeing, UKPost, AmEx?, Diamler-Chrysler?


TME
HP Openview
Tiv Conn
SNMP Conn
Reg Svc
Rel. Svc
WBEM
Manageability Services Broker
Jmx MR
JMX Adapt
Jmx MR
AppA MO
AppB MO
Aic mr
AIC Adapt
Aic mr
Java
C/C++

Design Goals

- Define very lightweight Broker
- Allow dynamic pluggable services
- Define minimal set required services
- Broker and component location independence
- Must be able to manage complex, distributed applications including corba based, eBusiness (app server based), and b2b applications.

Design Goals

- The same interfaces should be able to be used for feeding application specific managment system as well as any interested enterprise managment systems
- All calls/messages to the broker should be sent to the broker without any knowledge by the instrumentor that a service will ultimately satisfy the call. It is the brokers responsibility to map the call to the correct service to handle it.
- Might have multiple of the same services registered.

Design Goals

- Instrumentation Interface:
  - EASY to understand and use by application developers of below average skill.
  - code that needs to be inserted into the application must be generateable by IDE's and wizards (DII type interfaces make this easier).
  - should support CIM Schema based management objects as well as schemaless management objects.
  - should be easily extensible.

Affinity

- Need a standard 'default' interface - XML over HTTP
- Language: Need a way to 'negotiate' to communicate in a language between two components of the same language - Java or C, etc.
- Location: Need a way to 'negotiate' remote communication mechanism - In/out process, RMI, Socket based, Corba based, message based, etc.
- Schema: Allow to 'negotiate' if this is a schema or schemaless communication

Required Standard Services

- Service Management
- Instance Management
- Registration
- MetaData
- Delegation
- Relationships
- Query Static
- Query Dynamic
- Events

Required Standard Services

- Service Mangement (NEW) (Broker)
  - addService(string serviceName, string serviceInterfaceName, object NewService)
  - removeService(string serviceName)
  - queryService(string serviceInterfaceName)

Required Standard Services *(continued)*

- Instance Management (WBEM)
  - createInstance (object NewInstance)
  - getInstance(string instanceName, boolean localOnly)
  - deleteInstance(string instanceName)
  - modifyInstance(NamedObject modifiedInstance)
  - enumerateInstances(string ClassName, boolean LocalOnly, boolean DeepInheritance)
  - enumerateInstanceNames(string ClassName)

Required Standard Services *(continued)*

- Registration (NEW)
  - Register an existing object as the management object. Registrar may retain a handle to it, may be local or remote.
  - register(object newInstance, string instanceName)
  - unregister(string instanceName)

Required Standard Services *(continued)*

- MetaData (WBEM)
  - qualifierDecl getQualifier(string QualifierName)
  - setQualifier(qualifierDecl QualiferDeclaration)
  - deleteQualifier(string QualifierName)
  - qualifierDecl[] enumerateQualifiers ()

Required Standard Services *(continued)*

- Delegation (WBEM)
  - propertyValue getProperty(instanceName InstanceName, string PropertyName)
  - setProperty(instanceName InstanceName, string PropertyName,
  - propertyValue NewValue)
  - propertyValue[] enumerateProperties() (NEW)
  - (New) invokeMethod(string instanceName, string MethodName, object[] methodParms)

Required Standard Services *(continued)*

- Relationships (WBEM)
  - objectWithPath[] associators(objectName ObjectName, string AssocClass,string ResultClass, string Role, string ResultRole)
  - objectWithPath[] associatorNames (objectName ObjectName, string AssocClass,string ResultClass, string Role, string ResultRole)
  - objectWithPath[] references(objectName ObjectName, string ResultClass, string Role)
  - objectPath[] referenceNames(objectName ObjectName, string ResultClass,string Role)

Required Standard Services ***(continued)***

- Query Static (WBEM - query on static information only)
  - object[] execQuery(string QueryLanguage, string Query)
- Query Dynamic (WBEM - allows query on attribute values)
  - object[] execQuery(string QueryLanguage, string Query)

Required Standard Services *(continued)*

- Event Delivery (WBEM)
  - publishEvent(Event)
  - subscribeEvent(Query)
  - unsubscribeEvent(Query)
  - data: eventID, severity, timestamp, text, sequence#, originator

Optional Standard Services

- Naming (now standard?)
- Lookup (optional, but first rel.)
- Discovery
- Schema Service (optional?, first rel.)
- Application Lifecycle
- Transactions (optional, first rel.
- Collections
- Policy

Optional Standard Services

- Internal
  - Bootstrap (Internal)
  - Persistence (Internal)
  - Caching (Internal)
  - Security (Internal)
  - Request Forwarding (Internal)
- Application
  - Monitoring/Thresholding (App)
  - Logging (App)
  - Reporting (App)
  - Scheduling (App)

Optional Standard Services

- Naming
  - boolean checkName(string instanceName|className|serviceName |serviceInterfaceName,enumeration {instance|class|service|serviceInterface})
    - getName returns a valid Name for the component, if a proposed name is passed in it returns the same name if its was unique or a new or modified name if it was not unique or valid.
  - string getName()
  - string getName(string instanceName|className|serviceName| serviceInterfaceName,enumeration {instance|class|service|serviceInterface})

Optional Standard Services

- Lookup (NEW)
  - find(broker|instanceName|className|componentName|serviceName|ManagedResourceName|etc)
  - find(namePattern,componentType,domain)
  - advertise(broker|instanceName|className|componentName|serviceName|ManagedResourceName|etc)
- Discovery (NEW)
  - Is this a findAll discovery or a listenForNew discovery? both? of Brokers? of Manageable Resources? Both?

Optional Standard Services

- Schema Service (WBEM)
  - createClass(object NewClass)
  - modifyClass (NamedObject modifiedClass)
  - getClass (string className, boolean localOnly)
  - deleteClass (string className)
  - enumerateClasses(string ClassName, boolean DeepInheritance, boolean LocalOnly)
  - enumerateClassNames(string ClassName, boolean DeepInheritance)

Optional Standard Services

- Application Lifecycle (NEW)
  - start(object[] options)
  - stop(object[] options)
  - status(object[] options)
- Transactions (NEW)
  - startTransaction(transactionID)
  - endTransaction(transactionID)

Optional Standard Services

- Collections (NEW)
  - dynamic collection (query based) issues events to subscribers when members are added/deleted. The collection listens for lifecycle events from the broker.
  - createCollection(queryStatement)
  - createCollection(object[] instanceList)
- Policy (NEW? Based on WBEM?)
  - setPolicy(policyRule)
  - getPolicy(policyRule)

Optional Standard Services

- Bootstrap (NEW)
  - initFile(string fileName)
  - instantiateObjects(object[] objectList)
  - This would include instances, classes, or services.
- Persistence (NEW)
  - load()
  - store()
- Caching (NEW)
  - cacheValue()
  - getFromCache()
  - setCachePolicy()

Optional Standard Services

- Security (NEW)
- Request Forwarding (NEW)
  - forwardRequest(target)
- Monitoring/Thresholding (NEW)
  - poll()
  - ping()
  - evaluateThreshold()

Service Capabilities Advertising

- Basic Read: get/enumerate methods of Instance Service, Schema Service, and Delegation Service
- Basic Write: Basic Read + Delegation Service
- Schema Manipulation: Instance Manipulation + Schema Service
- Instance Manipulation: Basic Write + InstanceService
- Association Tranversal: Basic Write + Relationships
- Query Execution: Basic Write + QueryStatic
- Qualifier Declaration : Schema Manipulation + MetaData Service