Dynamic Deployment of IIOP-Enabled Components in the JBoss Server

Francisco Reverbel
CS Department, University of São Paulo
Bill Burke
JBoss, Inc.
Marc Fleury
JBoss, Inc.

Outline

- Introduction
- JBoss Background
  - JMX, Service Components, Meta-Level Architecture
- The IIOP Module
- Proxy Factories
- IIOP Invoker and IOR Factory Internals
- Related Work
- Concluding Remarks

Introduction

- Introduction
- JBoss Background
- The IIOP Module
- Proxy Factories
- IIOP Invoker and IOR Factory Internals
- Related Work
- Concluding Remarks

Software Components in the JBoss Server

- Application components
  - Server-side parts of distributed applications
- Middleware components
  - Provide middleware (system-level) services to application components

Both kinds of components can be dynamically deployed into a running server

Component Models

- Application components follow J2EE standards
  - EJB model: for business components (“enterprise beans”) whose methods can be invoked either by remote clients or by local clients
  - Servlet/JSP model: for “web components”
- Middleware components employ an extension of the **JMX model**
  - This extension is JBoss-specific

IIOP-Enabled EJB Components

- IIOP support is mandatory for EJB compliance

JBoss supports IIOP in a dynamic way:

- IIOP support is a feature that can by dynamically added to a running server
- IIOP-enabled EJBs are dynamically deployable components themselves
  - No extra compilation steps for IIOP stub/skeleton generation
  - JBoss uses reflective techniques instead

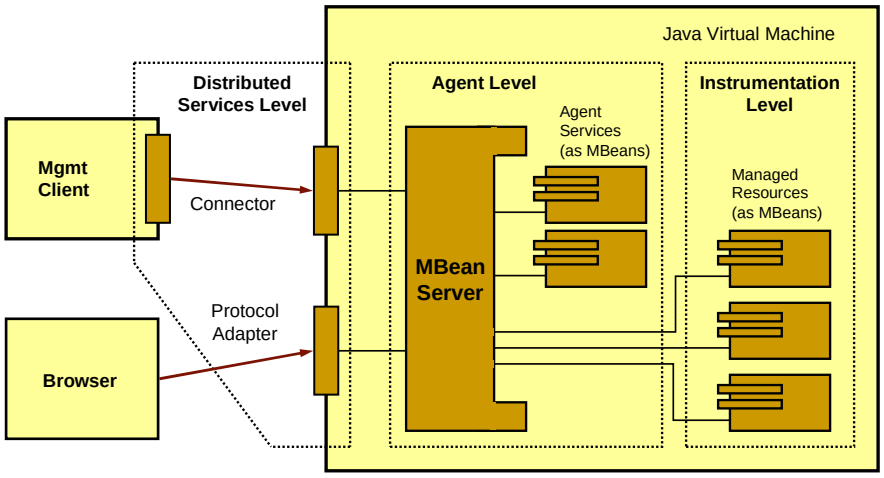
Hot Deployment

- Users can deploy IIOP-enabled EJBs into a running server simply by dropping deployment units (such as EJB-JAR files) in the server's deployment directory
- These deployment units contain no IIOP stubs and no IIOP skeletons
- They contain no JBoss-specific classes (such as EJB container or CORBA servant classes)

A change to a JBoss-specific deployment descriptor is all that is needed to convert a non-IIOP-enabled deployment unit into an IIOP-enabled one

JBoss Background

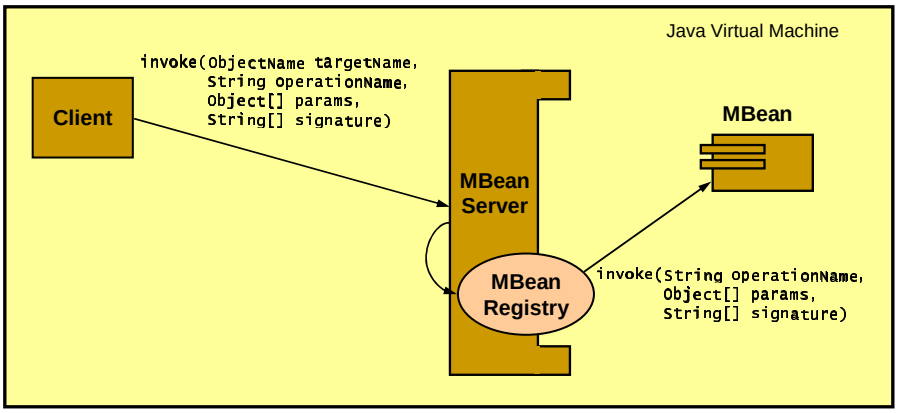
JMX Architecture



The JMX component model

- The instrumentation and agent levels define an in-process component model
- The MBean server provides a registry for JMX components (MBeans)
- Each MBean is assigned an object name that is unique in the context of the MBean server
- Clients use object names to refer to MBeans
- MBean operation invocations always go through the MBean server

Method invocation on a dynamic MBean



- The client holds no direct Java references to the MBean
- It does not need to know the MBean's Java class/interfaces
- This very simple arrangement favors adaptation

Issues not covered by JMX

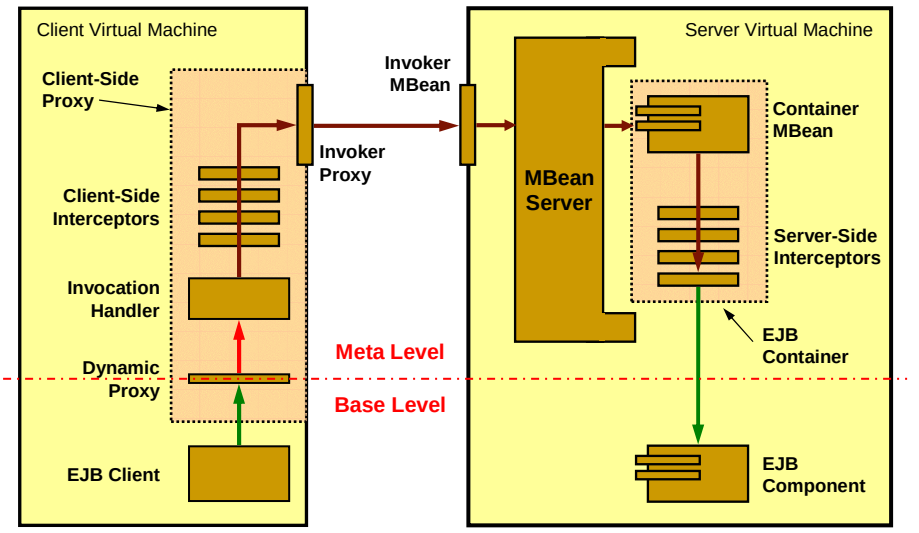
- Dependencies between MBeans
- Service lifecycle
- Packaging and deployment of MBeans

JBoss addresses these issues with the notions of service MBean and deployable MBean

The JBoss model for middleware components

- Service MBeans:
  - MBeans whose management interfaces include service lifecycle operations
  - Also called service components
- Deployable MBeans:
  - Service MBeans packaged according to EJB-like conventions, in deployment units called service archives (SARs)
    - A SAR includes a service descriptor, an XML file that conveys information needed at deployment time
  - Also called deployable services
    - They are a JBoss specific extension to JMX

Meta-level architecture for EJB: the case of a non-IIOP client



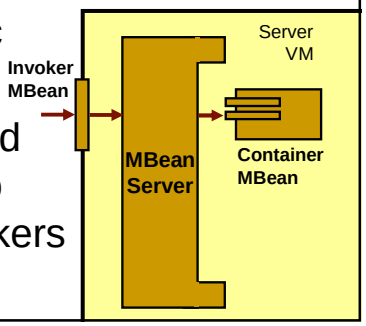
Reified method invocations

```
class Invocation {
  Object objectName;
  java.lang.reflect.Method method;
  Object[] args;
  InvocationContext invocationContext;
  ...
}
```

- No interface along the reified invocation path depends on base-level application types

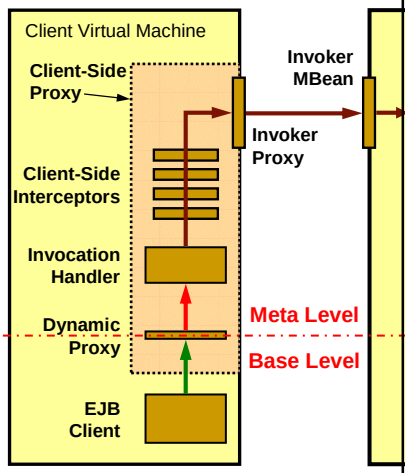
Remote invocation architecture for non-IIOP clients (1 of 2)

- EJB containers expose a type-independent `invoke` operation
- Protocol-specific invoker MBeans make this operation accessible to remote clients through various protocols (JRMP, HTTP, SOAP)
- Client-side stubs are dynamic proxy instances
- They convert calls to the typed interfaces seen by clients into `invoke` calls on remote invokers



Remote invocation architecture for non-IIOP clients (2 of 2)

- Each client-side proxy has an invocation handler that performs remote calls on a given invoker, over the protocol supported by the invoker
- Client side proxies and their invocation handlers are instantiated by the server and dynamically sent out to clients as serialized objects



JBoss invokers

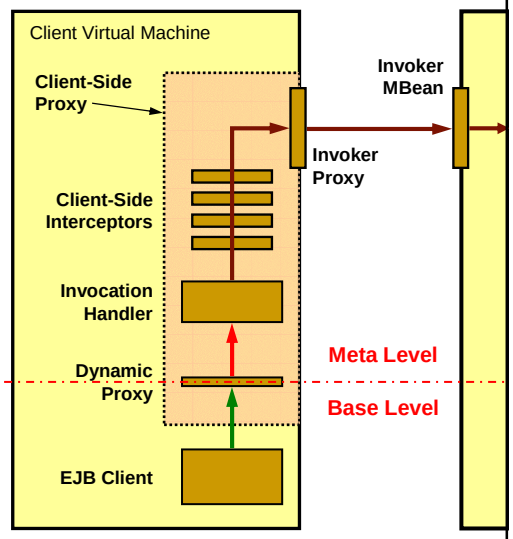
- Deployable MBeans that act as protocol-specific gateways to multiple EJB containers
- Interface exposed by the JRMP invoker:

```
interface Invoker extends javax.rmi.Remote {
    String getServerHostName();
    Object invoke(Invocation invocation);
}
```

- Other non-IIOP invokers implement either this interface or very similar ones

Invoker proxies

- A client-side proxy (or, more precisely, its invocation handler) must "know a remote invoker"
- This knowledge is protocol-specific
- It is encapsulated within an invoker proxy



The IIOP Module

- Introduction
- JBoss Background
- The IIOP Module
- Proxy Factories
- IIOP Invoker and IOR Factory Internals
- Related Work
- Concluding Remarks

The IIOP module consists of...

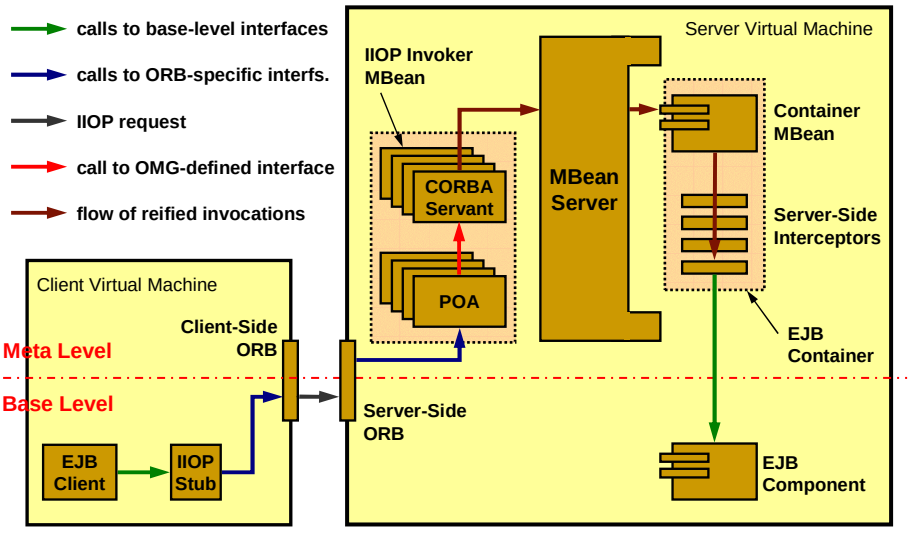
… three deployable MBeans:

- `CORBAOrbService`
  - Thin wrapper around a third-party IIOP engine
  - JacORB is the default IIOP engine in the JBoss distribution
- `CorbaNamingService`
  - In-process CORBA naming service
  - Reuses (through inheritance) code from the JacORB naming service
- `IIOPInvoker`

The IIOP invoker

- Unlike all other invokers, it does not follow the JBoss “invoker pattern”
- IIOP is treated as an special case
  - Reason: interoperability with CORBA clients written in other languages
  - The “invoker pattern” leads to an IIOP invoker that expects IIOP requests with the verb `invoke` in their operation fields
  - But IDL-generated stubs send out IIOP requests whose operation fields contain application-specific verbs

Meta-level architecture for EJB: the case of an IIOP client



Proxy Factories

- Introduction
- JBoss Background
- The IIOP Module
- Proxy Factories
- IIOP Invoker and IOR Factory Internals
- Related Work
- Concluding Remarks

EJB references

- An EJB reference is either a reference to an `EJBObject` or a reference to an `EJBHome`
  - EJB containers have the responsibility of creating such references, which in various situations they pass to EJB clients or to EJB implementations
- JBoss supports two general kinds of EJB references:
  - Java references to dynamic proxy instances, which are sent out to other VMs as serialized Java objects
  - CORBA references, passed across process boundaries in the standard IOR format

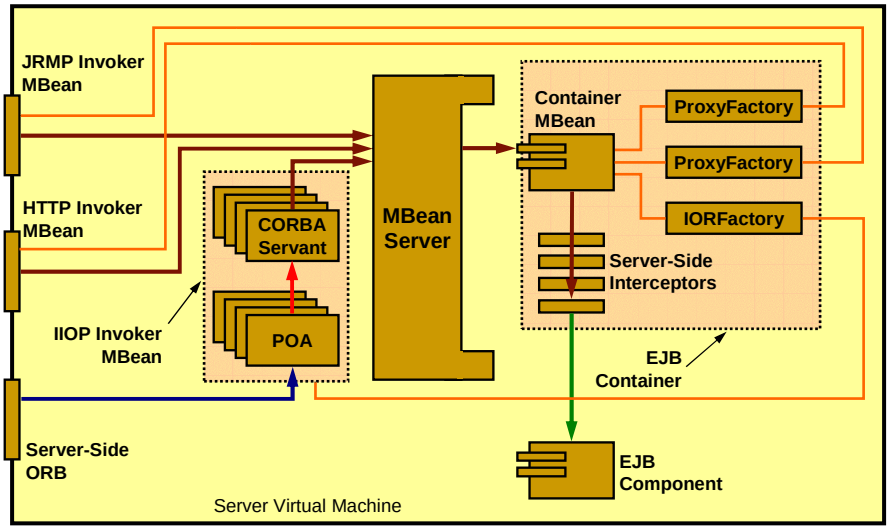
The proxy factory interface

```
public interface EJBProxyFactory
    extends ContainerPlugin {
  ...
  Object getEJBHome();
  Object getStatelessSessionEJBObject();
  Object getStatefulSessionEJBObject(
                          Object sessionId);
  Object getEntityEJBObject(
                          Object primaryKey);
  Collection getEntityCollection(
                     Collection primaryKeys);
}
```

Implementations of the EJBProxyFactory interface

- `ProxyFactory`
  - Instances of this class create dynamic stubs that talk to remote invokers over various protocols
- `ProxyFactoryHA`
  - Instances of this class create dynamic stubs used in clustered JBoss environments
    - The “HA” suffix stands for “high availability”
- `IORFactory`
  - Instances of this class create CORBA references

Relationship between invokers, proxy factories and containers



Proxy factory configurations

- EJB deployer MBean
  - Handles the deployment of EJB components
  - Reads container configurations from XML files and creates containers
- Container configuration
  - Has all the information the EJB deployer needs to create a container MBean, its plug-ins and its interceptors
  - Includes information on the container's proxy factories
    - One proxy factory configuration per protocol

Information in a proxy factory configuration

- The proxy factory class (a Java class)
- The invoker MBean (the protocol!)
- Additional parameters, depending on the proxy factory class
  - In the case of non-IIOP access, XML elements in the proxy factory configuration fully specify the chain of client-side interceptors
  - No similar elements exist in the configuration of a proxy factory for IIOP access (`IORFactory`)
    - Interceptors instantiated at the server side would not make sense for non-Java clients

Global configuration file

- Default container configurations for standard kinds of EJBs:
  - Stateless session beans
  - Stateful session beans
  - Entity beans
  - Message-driven beans

  These configurations support RMI over JRMP
- Alternative container configurations:
  - IIOP support
  - clustering
  - ...

Local configuration file

- JBoss-specific deployment descriptor
  - Optionally included with a given EJB
- May refer to an alternative container configuration by its name
  - Perhaps to specify other protocol, such as IIOP
- May fully define a new container configuration
- May use a predefined configuration and enhance it with aditional proxy factory configurations, also taken from the global configuration file
  - EJB accessible via multiple protocols

IIOP Invoker and IOR Factory Internals

- Introduction
- JBoss Background
- The IIOP Module
- Proxy Factories
- IIOP Invoker and IOR Factory Internals
- Related Work
- Concluding Remarks

CORBA servants

- Two CORBA servants per IIOP-enabled EJB
  - home servant and bean servant
  - They must implement IDL interfaces not known in advance!
- Our servants use the stream-based ORB API
  - Each servant knows the object name of the target container MBean
  - Each servant has marshalling knowledge specific to the IDL interface it implements
    - Map from IDL operation names to `SkeletonStrategy` objects

POA usage

- Our choice of lifetimes for CORBA references:
  - Session bean instances → transient references
  - Entity bean instances → persistent references
  - EJB homes → persistent references
- In terms of POAs:
  - Session bean servants are registered with POAs with the TRANSIENT lifespan policy
  - Entity bean and home servants are registered with POAs with the PERSISTENT lifespan policy
  - These POAs may be per-servant or shared
    - Default: per-servant POAs created at deployment time

Lazy generation of RMI/IIOP stub classes (1 of 3)

- An `IORFactory` can be optionally associated with a web class loader that generates RMI/IIOP stub classes in a lazy way
  - `WebCL` is a subclass of URLClassLoader
    - It has a method `byte[] getBytes(Class clz)`
  - When a `WebCL` is asked to load `Foo_Stub`, it
    - performs Java introspection on interface `Foo`
    - applies the Java to IDL mapping on `Foo`
    - uses code generation techniques to create a byte code array with the class file for `Foo_Stub`
      - keeps this array in a hash map, to be returned by `getBytes`
    - creates a `Class` instance from the byte code array

Lazy generation of RMI/IIOP stub classes (2 of 3)

- An in-process web server allows remote clients to download classes via HTTP
- A client downloads a class file by issuing an HTTP request to a resource name that includes
  - the id of a `WebCL` instance that can load the class
  - the full name of the class file
- When the web server receives such a request, it uses the `WebCL` given by the resource name
  - to create a `Class` object
  - to retrieve the corresponding class file

Lazy generation of RMI/IIOP stub classes (3 of 3)

Default `IORFactory` configuration:

- Every `IORFactory` has its `WebCL` instance
- An `IORFactory` includes information on its web class loader in each IOR it generates
  - The IOR has a tagged component that specifies a codebase URL for stub downloading
  - The path component in this URL identifies the web class loader of the `IORFactory` that created the IOR

Related Work

- Introduction
- JBoss Background
- The IIOP Module
- Proxy Factories
- IIOP Invoker and IOR Factory Internals
- Related Work
- Concluding Remarks

Some related middleware systems

- Flexinet (Hayton et al)
  - Flexible remote invocation paths
  - Dynamic stubs
- OpenCOM (Clarke et al)
  - Lightweight component model
  - Based on a subset of COM
  - Dependence management, reconfiguration, method call interception
- Hadas (Ben-Shaul et al)
- Fractal component model (Bruneton et al)

Commercial J2EE servers

- Most use compilation-based approaches
- IONA's ART framework
  - relies on the chain of responsibility pattern
  - basis for various middleware products, including a J2EE server
- Nearly all commercial offerings require vendor-specific EJB container or CORBA servant classes to be generated as part of the deployment process
- In the case of IIOP-enabled EJBs, all other servers require extra compilation steps for stub and skeleton generation

Concluding Remarks

JBoss/IIOP...

- Supports IIOP-enabled components without sacrificing the levels of flexibility, developer friendliness and ease of use that JBoss had already reached in the non-IIOP case
- Strong reliance on reflection to make EJB deployment easy

Dynamic deployment issues...

- Instantiation of CORBA servants for IDL interfaces not known in advance
- Creation of POAs at deployment time
- Lazy generation of RMI/IIOP stubs

We believe that our techniques are equally applicable to CCM servers and to other highly dynamic component environments.

Thank you for your attention!

- Q & A ...

Francisco Reverbel
reverbel@ime.usp.br

Bill Burke
bill@jboss.org

Marc Fleury
marc@jboss.org

http://www.jboss.org

Backup Slides

Dynamic proxy usage

- A dynamic proxy is an object adapter that converts the type-independent interface of its invocation handler into a list of interfaces specified at runtime
- Dynamic proxies bridge the gap between the interfaces that are application-specific and those exposed by middleware components

The dynamic stub idiom

- In Java RMI:
  - Serializable types are normally passed by value
  - Remote types are normally passed by reference
  - What about a remote object that is also serializable?
    - If it has not been exported through the RMI system, then it will be passed by value (in serialized form)
    - This allows the creation of custom stubs, which interact over a custom protocol with the remote objects they represent
- JBoss uses dynamic proxies as custom stubs
  - The dynamic proxy implements application interfaces
  - Its customized part is the invocation handler

Generalized EJB containers

- A container MBean is created when an EJB is deployed
- It provides middleware services to its EJB
  (instance pooling, instance caching, persistence, security, transactions...)
- … by merely aggregating aspects that do the real work
- Container configurations (XML files)
  - For standard kinds of EJBs
  - For JBoss-specific extensions
  - Customized containers for generalized EJBs

JBoss...

- Demonstrates that application servers can be built out of dynamically deployed components that provide middleware services to application components
- Brings reflective middleware to the world of mainstream computing