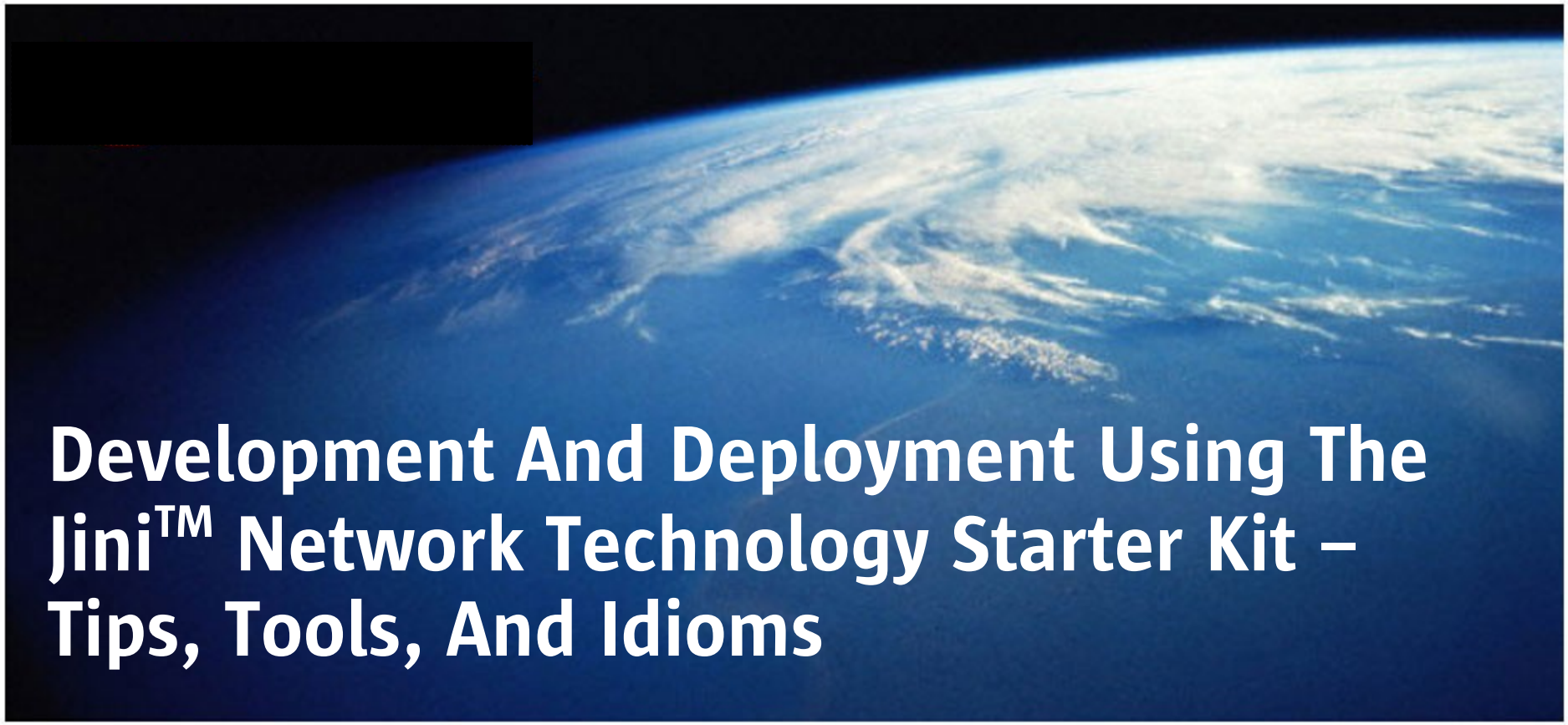


Development And Deployment Using The Jini™ Network Technology Starter Kit – Tips, Tools, And Idioms

Brian Murphy

Sun Microsystems, Inc.

Sun microsystems
We make the net work.

Presentation Agenda – Part I

- Introduction
- Patterns for development
  - For services and clients
- A simple example service
- A simple example client
- Patterns and tools for deployment
  - Preferred classes
  - ClassDep tool
  - Packaging for deployment
  - Configuration for JRMP and Jini ERI (JERI)
  - Security policy (JRMP and JERI)

Presentation Agenda – Part II

- Deploying for secure remote communication using Jini ERI/JSSE
  - Service and client configuration
  - Service and client policy
- Debugging
  - Some random tips
  - DebugDynamicPolicyProvider
- Deploying the example with network security and JERI/JSSE

Introduction

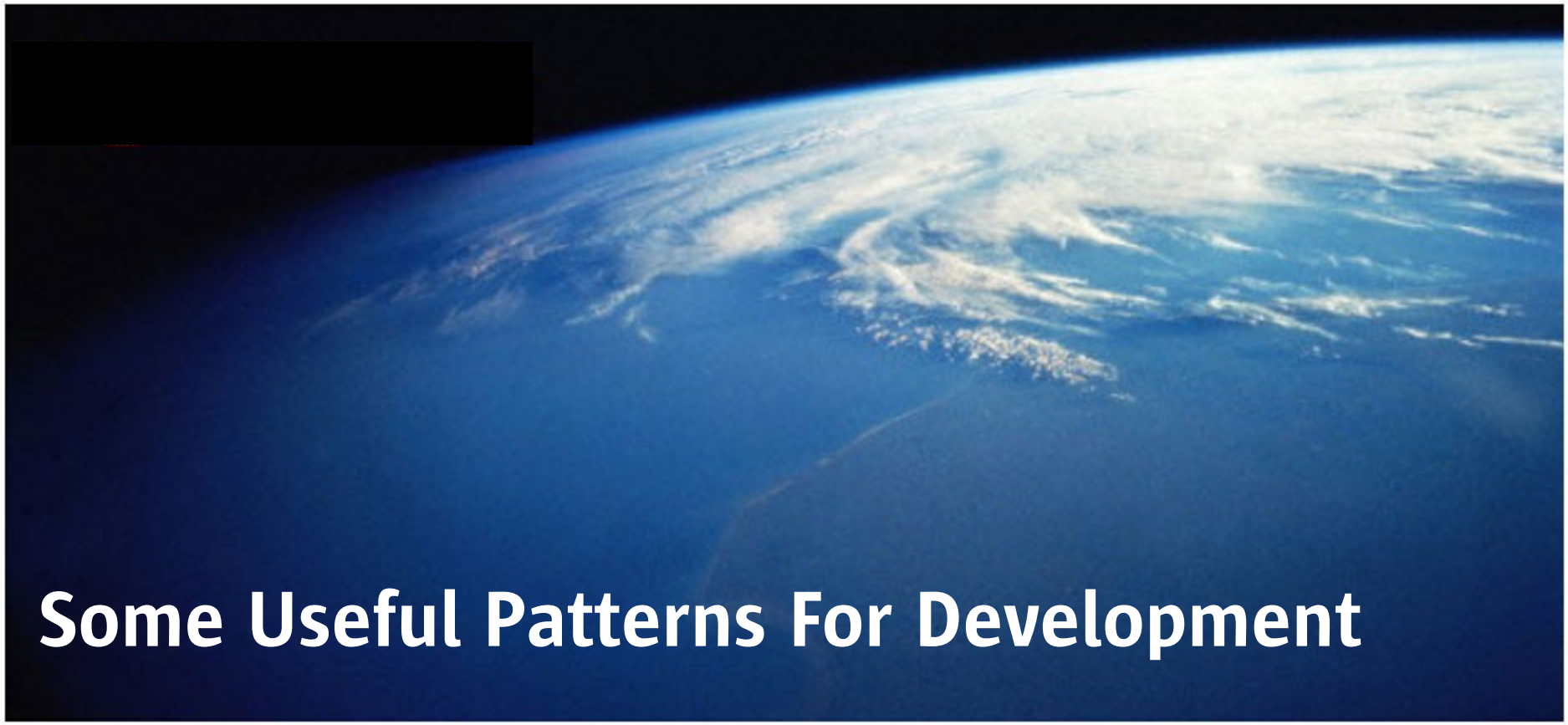
Sun microsystems
We make the net work.

Recall: Jini Technology Is

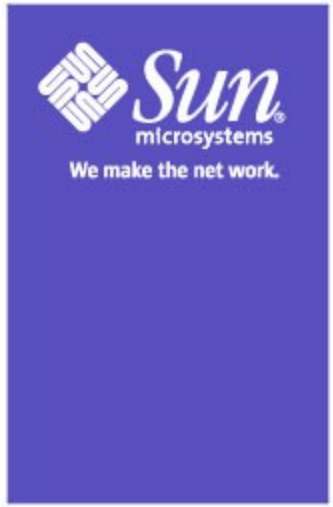
- A set of specifications
  - Core specifications – the “essence”
  - Standard, non-core specifications (`net.jini`)
  - Services and utilities
- A programming model
- Contributed implementations (Jini Technology Starter Kit)
  - Service and utility implementations
- A community

The Jini Programming Model

- Discovery – bootstrapping
- Distributed leasing – resource reclaim
- Remote events – async notification
- Transactions – distributed consensus
- Lookup – resource acquisition
- Distributed shared memory (JavaSpaces)
- Comprehensive network security



Some Useful Patterns For Development



Common Service Pattern

- Retrieve configuration
- Establish Subject (login)
- Initialize (from configuration)
- Discover and join lookup service(s)
  - Manage renewal of received leases (tennant)
    - LeaseRenewalManager
  - JoinManager
- Service client requests
- Typically long lived

Questions To Ask

- Define the resource the service provides
  - Is the resource a leased resource?
  - Manage leases granted by the service
    - LandLord
    - Custom lease management code
- Does the service generate events?
  - Manage event registrations
    - 'EventManager' utility
    - Custom event management code
- Discover and use other services? (client)
- Should it be administrable?

Common Client Pattern

- Retrieve configuration
- Establish Subject (login)
  - Execute all code inside doAsPrivileged block
- Initialize (from configuration)
- Discover services – known interfaces
  - Discover lookup service(s)
  - Query lookup(s) and/or register for events
  - ServiceDiscoveryManager (SDM)
- Use the discovered service(s)
  - Manage renewal of received leases (tennant)
- Can be short lived or long lived

Exporters And Proxy Preparers

- If a service or client has a remote object (if it provides a proxy), then configure an exporter
  - Service backend
  - Remote admins
  - Listeners that receive remote events
- If a service or client receives a proxy then retrieve and apply proxy preparer
  - The helper utilities take care of a lot of proxy preparation details for you

Compilation

- Starter kit currently optimized for deployment (lib/*.jar)
  - Expand starter kit jar files to <classes> directory
    - jsk-platform.jar
    - jini-ext.jar
    - sun-util.jar
    - start.jar
  - Expand remaining jar files if you're going to recompile the starter kit itself
- Compile against <classes> directory
- But build for deployment

Common Service Structures

- Local, non-remote proxy
  - Runs locally in client's VM
- RMI “stub”
  - Manually generated at compile time (rmic)
  - Dynamic proxy – automatically generated
  - Remote methods only
- Smart proxy
  - Local and remote methods
  - Simple pattern (non-leased resource)
  - Registration pattern

Smart Proxy Pattern – Frontend

- ServiceInterface (non-Remote)
  - Well known public methods (service's resources)
    - What's advertised/searched on in lookup services
    - Specifies remote and non-remote methods
- ServiceProxy (primary)
  - Implements ServiceInterface
  - Administrable
- ServiceAdminProxy (secondary)
  - JoinAdmin
  - DestroyAdmin

Smart Proxy Pattern – Backend

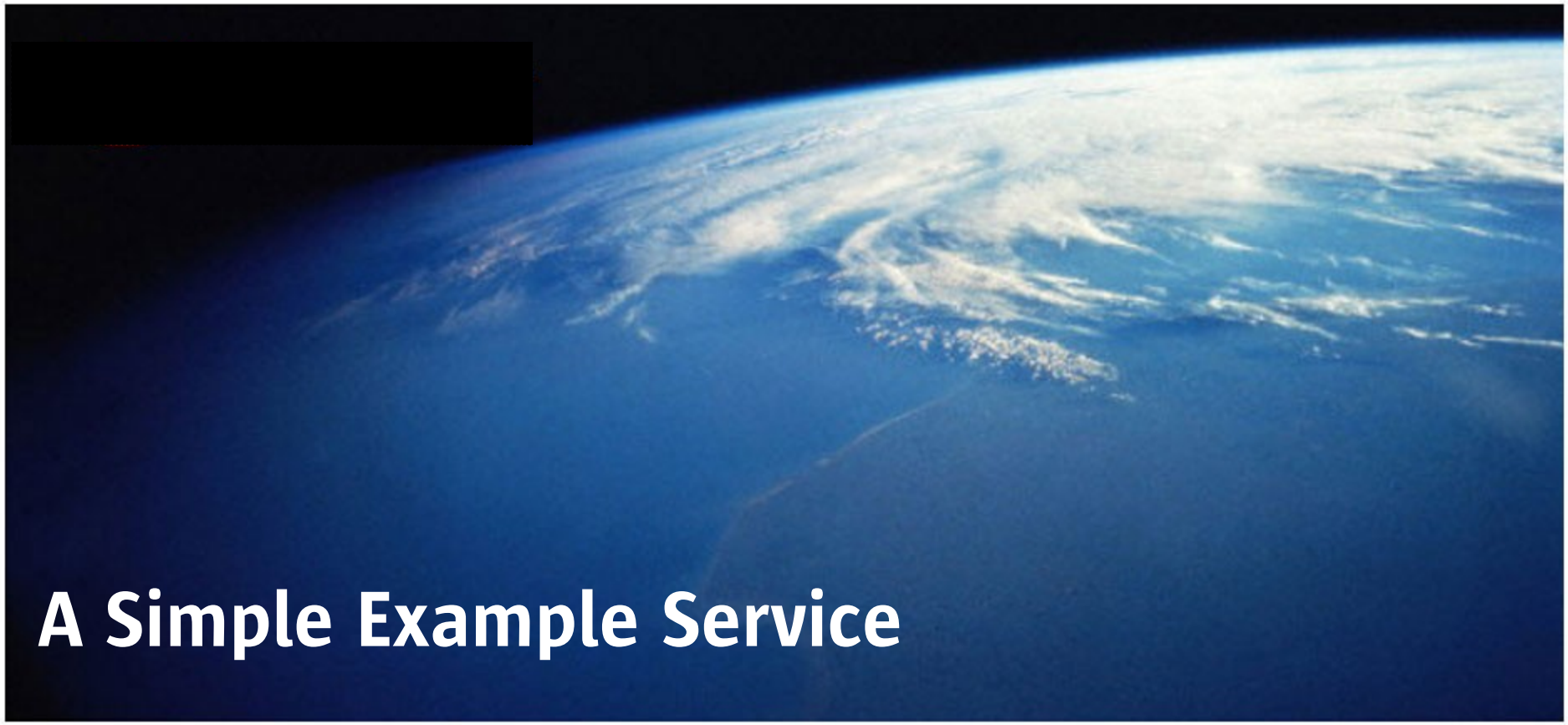
- RemoteServiceInterface
  - Defines private protocol between client-side proxies (frontend) and the server (backend)
  - Implements:
    - Remote
    - Administrable
    - JoinAdmin
    - DestroyAdmin
- ServiceImpl
  - Implements RemoteServiceInterface
  - The 'server' or service's 'backend'

The “Registration Pattern”

- ServiceInterface
  - Specifies a 'register' method
- ServiceRegistrationInterface
  - Returned by “register” method
  - Manage relationship with leased resource
- ServiceProxy
- ServiceAdminProxy
- ServiceRegistrationProxy
- RemoteServiceInterface
- ServiceImpl

Example: The Lookup Service

- ServiceRegistrar/RegistrarProxy
  - 3 resources: 2 leased, 1 not leased
    - register – resource = residency/advertisement
    - notify – resource = event mechanism
    - lookup – resource = service reference
- ServiceRegistration/Registration
  - Manage residency
- EventRegistration/EventRegistration
  - Manage registration with event mechanism
- *Admin/AdminProxy
- Registrar/RegistrarImpl
  - Backend implementation of resources & admin



A Simple Example Service

Sun microsystems
We make the net work.

Service Recipe

- Specify resource(s) provided by service
  - Leased or not leased?
- Specify public interface
- Define proxies
- Define remote interface
- Define service implementation
  - Boilerplate code
  - Service-specific code
  - Support code

Service Interface

- What clients know about and search on

```
package examples.sunw;

import java.rmi.RemoteException;

public interface Service {

    public double getIPO();

    public double getPrice()
                            throws RemoteException;
}
```

Smart Proxy Structure

- Fields
  - Inner proxy (reference to the backend server)
  - UUID for service equality
- Static factory and private constructor
  - Allows for RemoteMethodControl
- Methods required by the interfaces
- Methods for good “proxy behavior”
  - hashCode and equals
  - Serialization-related methods

Smart Proxy

```
package examples.sunw;
/* appropriate import statements */
class ServiceProxy implements Service, ReferentUuid,
                              Administrable, Serializable
{
    private static long serialVersionUID = 1L;
    final RemoteService innerProxy;
    final Uuid          proxyID;
    final double        ipo;
    public static ServiceProxy createServiceProxy
            (RemoteService innerProxy, Uuid proxyID, double ipo)
    {
        if(innerProxy instanceof RemoteMethodControl) {
            return new ConstrainableServiceProxy
                            (innerProxy, proxyID, ipo, null);
        } else {
            return new ServiceProxy(innerProxy, proxyID, ipo);
        }//endif
    }
    private ServiceProxy
            (RemoteService innerProxy, Uuid proxyID, double ipo)
    {
        this.innerProxy = innerProxy;
        this.proxyID = proxyID;
        this.ipo = ipo;
    }//end constructor
```

Methods From The Interfaces

```
..........

/* From public service interface */
public double getIPO() {
    return ipo;
}
public double getPrice() throws RemoteException {
    return innerProxy.getPrice();
}

/* From ReferentUuid interface */
public Uuid getReferentUuid() {
    return proxyID;
}

/* From Administrable interface */
public Object getAdmin() throws RemoteException {
    return innerProxy.getAdmin();
}
...........
```

For Good Proxy Behavior

```
..........

public int hashCode() {
    return proxyID.hashCode();
}
public boolean equals(Object obj) {
    return ReferentUuids.compare(this,obj);
}
private void readObject(ObjectInputStream s)
    throws IOException, ClassNotFoundException
{
    s.defaultReadObject();
    if(innerProxy == null) {
        throw InvalidObjectException("innerProxy null");
    }
    if(proxyID == null) {
        throw InvalidObjectException("proxyID null");
    }
}
private void readObjectNoData() throws IOException {
    throw InvalidObjectException("no data");
}
...........
```

Supporting Constraints

```
static final class ConstrainableServiceProxy
                                    extends ServiceProxy,
                                    implements RemoteMethodControl {
    private static long serialVersionUID = 1L;
    private MethodConstraints methodConstraints;
    /* Mapping: public Service --> private RemoteService */
    private ConstrainableServiceProxy(RemoteService innerProxy,
                                      Uuid proxyID,
                                      double ipo,
                                      MethodConstraints constrs) {
        super(constrainServer(innerProxy,constrs),proxyID,ipo);
        this.methodConstraints = constrs;
    }//end constructor
    public RemoteMethodControl setConstraints(...) {...}
    public MethodConstrains getConstraints(...) {...}
    private ProxyTrustIterator getProxyTrustIterator(...) {...}
    private readObject(ObjectInputStream s)
                    throws IOException, ClassNotFoundException {
        s.defaultReadObject();
        ConstrainableProxyUtil.verifyConsistentConstraints
              (methodConstraints, innerProxy, methodMapArray);
    }
}//end subclass ConstrainableServiceProxy

}//end class ServiceProxy
```

Admin Proxy

```
package examples.sunw;
/* appropriate import statements */
class AdminProxy implements JoinAdmin, DestroyAdmin,
                            ReferentUuid, Serializable {
    private static long serialVersionUID = 1L;
    final RemoteService innerProxy;
    final Uuid          proxyID;

    public static AdminProxy createAdminProxy
                        (RemoteService innerProxy, Uuid proxyID)
    {
        if(innerProxy instanceof RemoteMethodControl) {
            return new ConstrainableAdminProxy
                                    (innerProxy, proxyID, null);
        } else {
            return new AdminProxy(innerProxy, proxyID);
        }//endif
    }
    private AdminProxy(RemoteService innerProxy,Uuid proxyID){
         this.innerProxy = innerProxy;
         this.proxyID = proxyID;
    }//end constructor

    /* ConstrainableAdminProxy to support constraints */
```

Methods From The Interfaces

```
    ..........
    /* From JoinAdmin */
    public Entry[] getLookupAttributes()
                            throws RemoteException {
        return innerProxy.getLookupAttributes();
    }
    ..........

    /* From DestroyAdmin */
    public void destroy() throws RemoteException {
        innerProxy.destroy();
    }

    /* From ReferentUuid interface */
    public Uuid getReferentUuid() {
        return proxyID;
    }

    //Good behavior methods: hashCode, equals, etc.
}//end class AdminProxy
```

Smart Proxy Trust Verifier

```
package examples.sunw;
/* appropriate import statements */

final class ProxyVerifier implements Serializable, TrustVerifier {
    private static long serialVersionUID = 1L;
    private final RemoteMethodControl innerProxy;
    private final Uuid                proxyID;
    ProxyVerifier(RemoteService innerProxy, Uuid proxyID) {
        ..........
        this.innerProxy = (RemoteMethodControl)innerProxy;
        this.proxyID = proxyID;
    }//end constructor

    public boolean isTrustedObject
                    (Object obj, TrustVerifier.Context ctx)
                                        throws RemoteException
    {
        ..........
        /* If input proxy is (trust, content, and functionally)
         * equivalent to canonical inner proxy (same constraints)
         * and backend server's ID is equivalent to input proxy's
         * ID, return true; else return false.
         */
    }
}//end class ProxyVerifier
```

Backend Structure

- Remote service interface
  - Remote methods from public service interface
  - Remote admin methods
- Service implementation
  - Implements remote service interface
    - Service specific code
    - Admin code
  - Boilerplate code
  - Private support code

Backend Boilerplate

- ServiceStarter ready
  - LifeCycle – generally long-lived
- Initialization
  - Retrieve configuration
  - Login
  - Export
  - JoinManager
- Cleanup on exit

Remote Service Interface

```
package examples.sunw;

/* appropriate import statements */

interface RemoteService extends Remote,
          Administrable, JoinAdmin, DestroyAdmin
{

    public double getPrice()
                          throws RemoteException;

}
```

Service Implementation

```
package examples.sunw;
/* appropriate import statements */
class RemoteServiceImpl implements RemoteService {
    private LifeCycle     lifeCycle;//for ServiceStarter
    private Configuration config;
    private LoginContext  loginContext;
    private Uuid          proxyID;
    private ServiceID     serviceID;
    private Exporter      serverExporter;
    private RemoteService innerProxy;
    private ServiceProxy  outerProxy;
    private JoinManager   joinMgr;
    private double        ipo = 10.000;
    ..........
    RemoteServiceImpl(String[] args, LifeCycle lifeCycle)
                                                    throws Exception
    {
        this.lifeCycle = lifeCycle;
        try {
            init(args);
        } catch (Throwable e) {
            cleanupOnExit();
            //rethrow the exception to ServiceStarter
        }
    }//end constructor
    ..........
```

Methods From The Interfaces

```
..........

/* From remote service interface */
public double getPrice() { ... }

/* From Administrable interface */
public Object getAdmin() {
    return AdminProxy.createAdminProxy
                                (innerProxy,proxyID);
}

/* From JoinAdmin interface */
public Entry[] getLookupAttributes() {...}
public void addLookupAttributes(Entry[] attrSets){..}
..........

/* From DestroyAdmin interface */
public void destroy() { ... }

..........
```

Backend Boilerplate – init()

```
..........

private void init(String[] args) throws Exception {
    config = ConfigurationProvider.getInstance
             (args,
              (this.getClass()).getClassLoader() );
    try {
        loginContext
         = (LoginContext)Config.getNonNullEntry
                                (config,
                                 COMPONENT_NAME,
                                 "loginContext",
                                 LoginContext.class);
        initWithLogin(config, loginContext);
    } catch (NoSuchEntryException e) {
        doInit(config);
    }
}//end init
..........
```

Backend Boilerplate – initWithLogin()

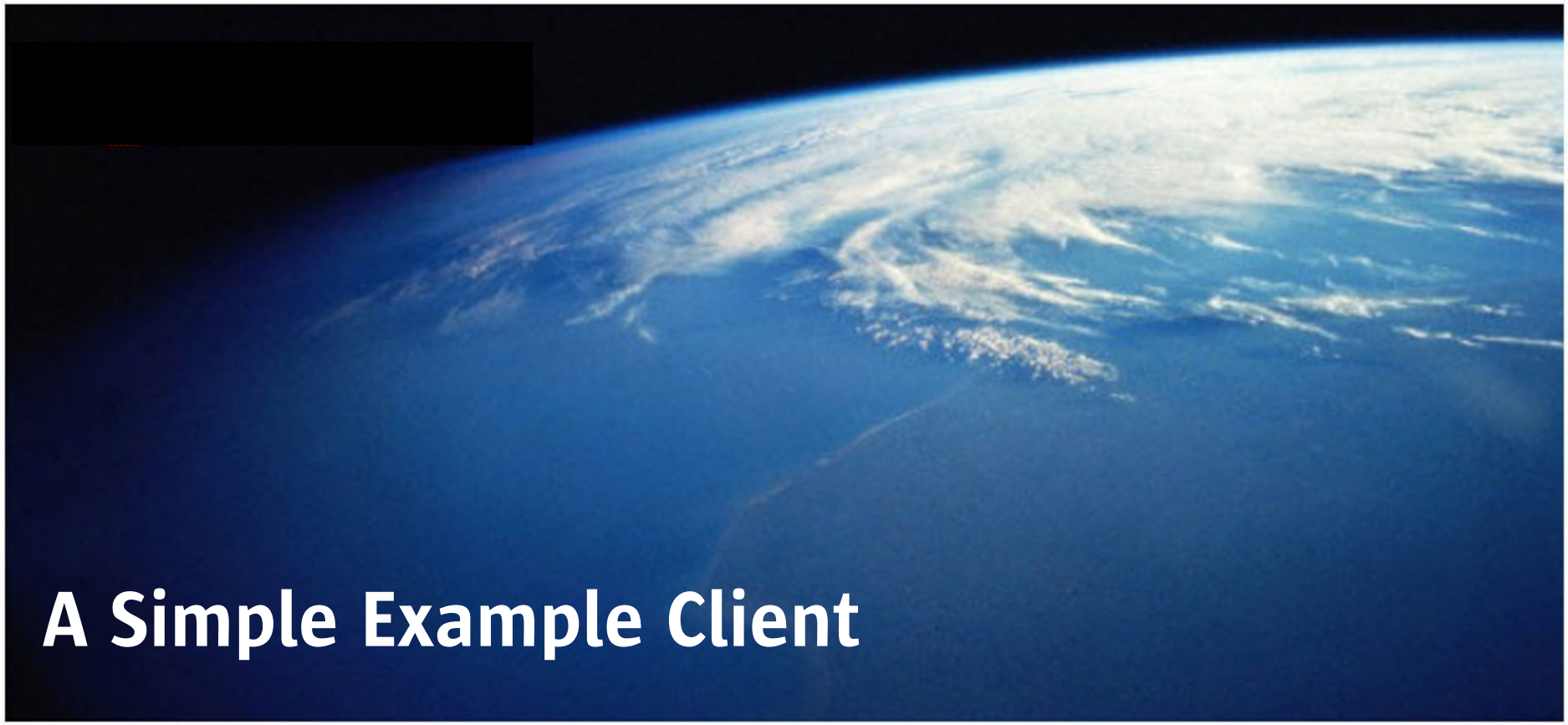
```
..........
private void initWithLogin(final Configuration config,
                           LoginContext loginContext)
                                        throws Exception
{
    loginContext.login();//authenticate configured Subject
    try {
        Subject.doAsPrivileged (
             loginContext.getSubject(),
             new PrivilegedExceptionAction() {
                 public Object run() throws Exception {
                     doInit(config);
                     return null;
                 }//end run
             },//end PrivilegedExceptionAction
             null
        );//end doAsPrivileged
    } catch (Throwable t) {
        /* Handle Throwable appropriately */
    }
}//end initWithLogin
..........
```

Backend Boilerplate – doInit()

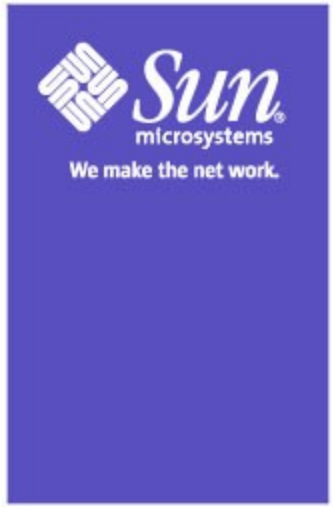
```
..........

private void doInit(final Configuration config)
                                throws Exception   {
    //set security manager if necessary
    proxyID = UuidFactory.generate();
    serviceID = new ServiceID
                (proxyID.getMostSignificantBits(),
                 proxyID.getLeastSignificantBits());
    //set default exporter
    serverExporter = (Exporter)Config.getNonNullEntry
                     (config, COMPONENT_NAME,
                      "serverExporter", Exporter.class,
                      defaultExporter);
    innerProxy = (RemoteService)serverExporter.export(this);
    outerProxy = ServiceProxy.createServiceProxy
                              (innerProxy, proxyID, ipo);
    //get additional config items
    joinMgr = new JoinManager(outerProxy, serviceAttrs,
                           serviceID, ldm, null, config);
}//end doInit
..........
```

Backend Boilerplate – Cleanup

```
    ..........
    private void cleanupOnExit() {
        if(innerProxy != null) {
            try {
                serverExporter.unexport(true);
            } catch(Throwable t) { /* ignore */ }
        }
        if(joinMgr != null) {
            try {
                joinMgr.terminate();
            } catch(Throwable t) { /* ignore */ }
        }
        if(ldm != null) {
            try {
                ldm.terminate();
            } catch(Throwable t) { /* ignore */ }
        }
    }//end cleanupOnExit
    ..........
}//end class RemoteServiceImpl
```



A Simple Example Client



Simple Client Structure

- Boilerplate code
  - Initialization
    - Retrieve configuration
    - Login – execute all code inside doAsPrivileged block
    - ServiceDiscoveryManager/LookupCache
    - Proxy preparer/ServiceItemFilter
  - Private support code
  - Cleanup on exit
- Discover and use targetted service(s)
- Different life cycle than service
  - Generally user controlled

Client Implementation

```
package examples.sunw;

/* appropriate import statements */

class Client {
    ..........
    private DiscoveryManagement     ldm;
    private ServiceDiscoveryManager sdm;
    private LookupCache             cache;
    ..........
    Client(String[] args) throws Exception {
        try {
            doInit(args);
        } catch(Throwable e) {
            cleanupOnExit();
            //rethrow the exception to main
        }
    }//end constructor
    ..........
```

Client Boilerplate – main()

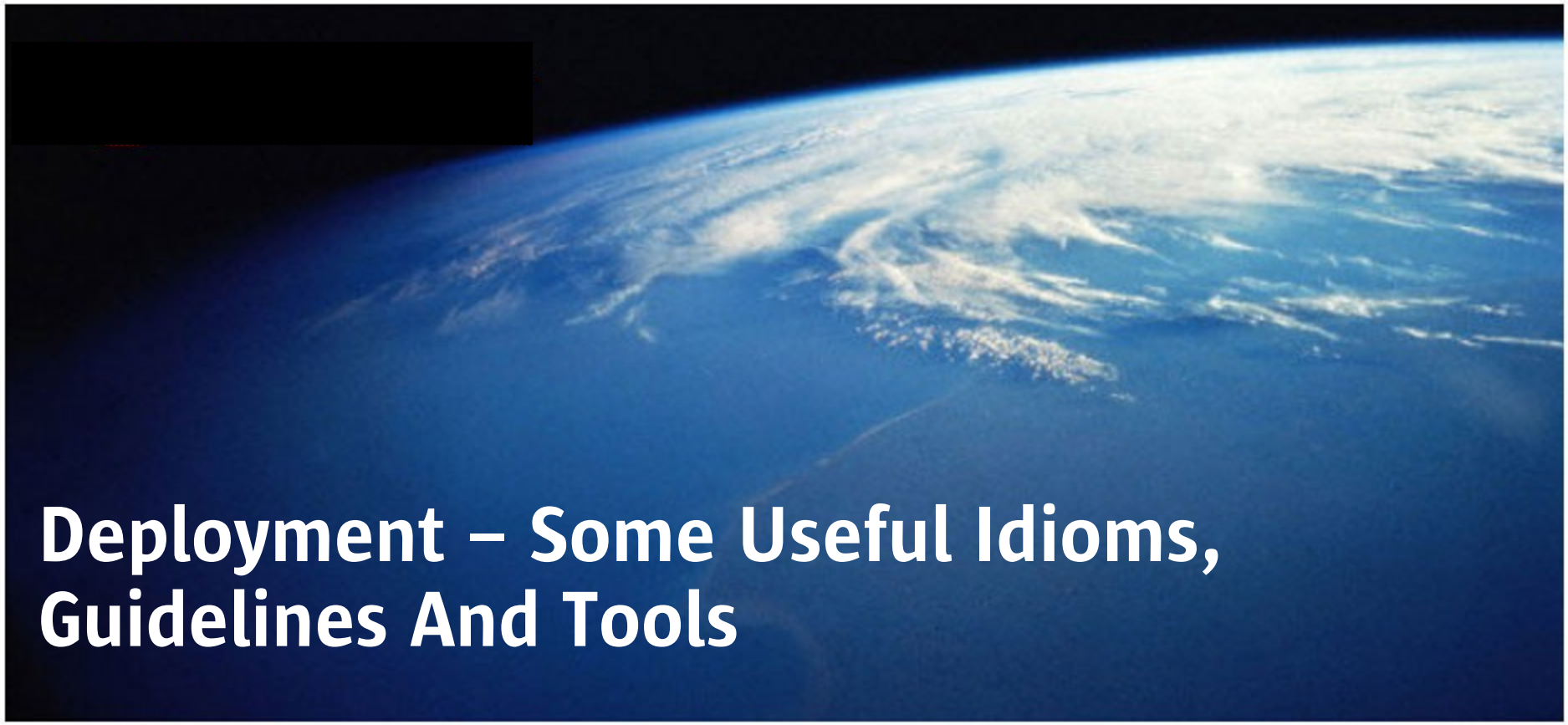
```
public void main(String[] args) throws Exception {
    final Configuration config;
    final LoginContext loginContext;
    try {
        config = ConfigurationProvider.getInstance(..);
        try {
            loginContext = (LoginContext)Config.getNonNullEntry(..);
            loginContext.login();
            Subject.doAsPrivileged (
                 loginContext.getSubject(),
                 new PrivilegedExceptionAction() {
                     public Object run() throws Exception {
                         Client thisClass = new Client(config);
                         thisClass.runClient();
                         return null;
                     }//end run
                 },//end PrivilegedExceptionAction
                 null
            );//end doAsPrivileged
        } catch(NoSuchEntryException e) {//run with no Subject
            Client thisClass = new Client(config);
            thisClass.runClient();
        }
    } catch(Throwable e) { e.printStackTrace(); }
}//end main
```

Client Boilerplate – doInit()

```
.........

private void doInit(final Configuration config)
                                              throws Exception
{
    //set security manager if necessary

    //get config items: servicePreparer, ldm

    sdm = new ServiceDiscoveryManager(ldm, null, config);
    Class[] serviceType
          = new Class[] {examples.sunw.Service.class};
    ServiceTemplate tmpl
          = new ServiceTemplate(null, serviceType, null);
    cache = sdm.createLookupCache
                       (tmpl,
                        new CacheFilter(servicePreparer),
                        new CacheListener());
}//end doInit

.........
```

Client Boilerplate – Cleanup

```
    ..........
    private void runClient() throws Exception {
        try {
            //discover lookups, discover service, use service
        } finally {
            cleanupOnExit();
        }
    }//end runClient

    private void cleanupOnExit() {
        if(sdm != null) {
            try {
                sdm.terminate();
            } catch(Throwable t) { /* ignore */ }
        }
        if(ldm != null) {
            try {
                ldm.terminate();
            } catch(Throwable t) { /* ignore */ }
        }
    }//end cleanupOnExit
    ..........

}//end class Client
```



Deployment – Some Useful Idioms, Guidelines And Tools

Sun microsystems
We make the net work.

Building For Deployment

- Preferred Classes
  - PREFERRED.LIST
    - service-dl.jar, client-dl.jar
- ClassDep
  - ClassDep output sent to 'jar' command
  - Not automatic, must pick 'roots'
    - Provide starting points ('roots')
    - Provide packages to include
    - Provide packages to exclude (platform)
    - Provide packages to hide
- Packaging
  - service.jar, service-dl.jar, client.jar client-dl.jar

Preferred Classes

- Bypasses 'load-local-first' mechanism
  - If a class is marked 'preferred', and it is available both locally and remotely, the remote version (the preferred version) will be loaded
  - If the class is only local, an exception occurs
- Mark 'preferred' anything the client side should not know about during development (compile time)
  - Client side generally knows about
    - Interfaces and utilities from net.jini
    - Administrative interfaces from com.sun.jini
    - Public interfaces from service(s) of interest

Preferred Lists – Idiom

- Create a directory named META-INF
  - <sunw>/META-INF
  - For anything that exports a remote object (proxies, leases, listeners, etc.)
- Create PREFERRED.LIST file in META-INF
  - Associated with the codebase
    - Included in the META-INF directory of the *-dl JAR file
  - Mark all preferred, then selectively backoff
    - com/sun/jini/start/*
    - com/sun/jini/admin/*
    - net/jini/-
    - Service-specific interface(s)

Preferred Lists – Examples

- <sunw>/manifest/service-dl/META-INF/PREFERRED.LIST

```
PreferredResources-Version: 1.0

Preferred: true

Name: com/sun/jini/start/*
Preferred: false

Name: com/sun/jini/admin/*
Preferred: false

Name: net/jini/-
Preferred: false

Name: examples/sunw/Service.class
Preferred: false
```

- <sunw>/manifest/client-dl/META-INF/PREFERRED.LIST

```
PreferredResources-Version: 1.0

Preferred: true
```

ClassDep – Service Classpath-A

```
java -cp <jini_home>/lib/tools.jar:<jdk_home>/lib/tools.jar
     com.sun.jini.tools.ClassDep
     -cp <jini_home>/classes:<sunw_home>/classes -files

     -in net.jini -in com.sun.jini -in examples.sunw

     -out net.jini.activation -out net.jini.config
     -out net.jini.constraint -out net.jini.core
     -out net.jini.export     -out net.jini.id
     -out net.jini.iiop       -out net.jini.io
     -out net.jini.jeri       -out net.jini.jrmp
     -out net.jini.loader     -out net.jini.security
     -out net.jini.url        -out com.sun.jini.discovery

     -hide net.jini -hide com.sun.jini

     examples.sunw.RemoteServiceImpl          [roots]
     examples.sunw.RemoteServiceImpl_Stub
```

ClassDep – Service Classpath-B

```
java -cp <jini_home>/lib/tools.jar:<jdk_home>/lib/tools.jar
     com.sun.jini.tools.ClassDep
     -cp <jini_home>/classes:<sunw_home>/classes -files

     -in net.jini -in com.sun.jini -in examples.sunw

     -out net.jini.activation -out net.jini.config
     -out net.jini.constraint -out net.jini.core
     -out net.jini.export     -out net.jini.id
     -out net.jini.iiop       -out net.jini.io
     -out net.jini.jeri       -out net.jini.jrmp
     -out net.jini.loader     -out net.jini.security
     -out net.jini.url        -out com.sun.jini.discovery

     -hide examples.sunw

     examples.sunw.RemoteServiceImpl          [roots]
     examples.sunw.RemoteServiceImpl_Stub
```

Classpath-A – service.jar

- 'sunw' classes to service.jar
  - 'jini' classes hidden, jsk-platform excluded

```
examples/sunw/AdminProxy.class
examples/sunw/AdminProxy$1.class
examples/sunw/AdminProxy$ConstrainableAdminProxy.class
examples/sunw/ProxyVerifier.class
examples/sunw/RemoteService.class
examples/sunw/RemoteServiceImpl.class
examples/sunw/RemoteServiceImpl$1.class
examples/sunw/RemoteServiceImpl$DestroyThread.class
examples/sunw/
    RemoteServiceImpl$LookupDiscoveryListener.class
examples/sunw/RemoteServiceImpl_Stub.class
examples/sunw/Service.class
examples/sunw/ServiceProxy.class
examples/sunw/ServiceProxy$1.class
examples/sunw/
    ServiceProxy$ConstrainableServiceProxy.class
```

Classpath-B – service.jar

- 'jini' namespace classes to service.jar
  - 'sunw' classes hidden, jsk-platform excluded

```
com/sun/jini/admin/DestroyAdmin.class
com/sun/jini/config/Config.class
com/sun/jini/lookup/entry/BasicServiceType.class
com/sun/jini/proxy/ConstrainableProxyUtil.class
com/sun/jini/start/LifeCycle.class
    ........
net/jini/admin/Administrable.class
net/jini/admin/JoinAdmin.class
net/jini/discovery/DiscoveryManagement.class
net/jini/lease/LeaseRenewalManager.class
net/jini/lookup/JoinManager.class
net/jini/lookup/entry/Name.class
net/jini/lookup/entry/ServiceType.class
    ........
```

ClassDep – Service Codebase-A

```
java -cp <jini_home>/lib/tools.jar:<jdk_home>/lib/tools.jar
     com.sun.jini.tools.ClassDep
     -cp <jini_home>/classes:<sunw_home>/classes -files
     -in net.jini -in com.sun.jini -in examples.sunw
     -out net.jini.activation -out net.jini.config
     -out net.jini.constraint -out net.jini.core
     -out net.jini.export     -out net.jini.id
     -out net.jini.iiop       -out net.jini.io
     -out net.jini.jeri       -out net.jini.jrmp
     -out net.jini.loader     -out net.jini.security
     -out net.jini.url        -out com.sun.jini.discovery
     -hide net.jini -hide com.sun.jini

     examples.sunw.ServiceProxy                  [roots]
     examples.sunw.AdminProxy
     examples.sunw.RemoteServiceImpl_Stub
     examples.sunw.ProxyVerifier
     net.jini.lookup.entry.Name
     net.jini.lookup.entry.BasicServiceType
```

ClassDep – Service Codebase-B

```
java -cp <jini_home>/lib/tools.jar:<jdk_home>/lib/tools.jar
     com.sun.jini.tools.ClassDep
     -cp <jini_home>/classes:<sunw_home>/classes -files
     -in net.jini -in com.sun.jini -in examples.sunw
     -out net.jini.activation -out net.jini.config
     -out net.jini.constraint -out net.jini.core
     -out net.jini.export     -out net.jini.id
     -out net.jini.iiop       -out net.jini.io
     -out net.jini.jeri       -out net.jini.jrmp
     -out net.jini.loader     -out net.jini.security
     -out net.jini.url        -out com.sun.jini.discovery
     -hide examples.sunw

     examples.sunw.ServiceProxy                [roots]
     examples.sunw.AdminProxy
     examples.sunw.RemoteServiceImpl_Stub
     examples.sunw.ProxyVerifier
     net.jini.lookup.entry.Name
     net.jini.lookup.entry.BasicServiceType
```

Codebase-A – service-dl.jar

- 'sunw' classes to service-dl.jar
  - jini' classes hidden, jsk-platform excluded

```
examples/sunw/AdminProxy.class
examples/sunw/AdminProxy$1.class
examples/sunw/
    AdminProxy$ConstrainableAdminProxy.class
examples/sunw/ProxyVerifier.class
examples/sunw/RemoteService.class
examples/sunw/RemoteServiceImpl_Stub.class
examples/sunw/Service.class
examples/sunw/ServiceProxy.class
examples/sunw/ServiceProxy$1.class
examples/sunw/
    ServiceProxy$ConstrainableServiceProxy.class
```

Codebase-B – service-dl.jar

- 'jini' namespace classes to service-dl.jar
  - 'sunw' classes hidden, jsk-platform excluded

```
com/sun/jini/admin/DestroyAdmin.class
com/sun/jini/lookup/entry/BasicServiceType.class
com/sun/jini/proxy/ConstrainableProxyUtil.class
net/jini/admin/Administrable.class
net/jini/admin/JoinAdmin.class
net/jini/entry/AbstractEntry.class
net/jini/lookup/entry/Name.class
net/jini/lookup/entry/ServiceControlled.class
net/jini/lookup/entry/ServiceInfo.class
net/jini/lookup/entry/ServiceType.class
```

Packaging The Service

- service.jar

```
cd <sunw_home>/classes
jar cvf <lib>/service.jar <Classpath-A output>

cd <jini_home>/classes
jar uvf <lib>/service.jar <Classpath-B output>
```

- service-dl.jar

```
cd <sunw_home>/classes
jar cvf <lib>/service-dl.jar -C <manifest>/service-dl/
   META-INF/PREFFERRED.LIST <Codebase-A output>

cd <jini_home>/classes
jar uvf <lib>/service-dl.jar <Codebase-B output>
```

ClassDep – Client Classpath-A

```
java -cp <jini_home>/lib/tools.jar:<jdk_home>/lib/tools.jar
     com.sun.jini.tools.ClassDep
     -cp <jini_home>/classes:<sunw_home>/classes -files

     -in net.jini -in com.sun.jini -in examples.sunw

     -out net.jini.activation -out net.jini.config
     -out net.jini.constraint -out net.jini.core
     -out net.jini.export     -out net.jini.id
     -out net.jini.iiop       -out net.jini.io
     -out net.jini.jeri       -out net.jini.jrmp
     -out net.jini.loader     -out net.jini.security
     -out net.jini.url        -out com.sun.jini.discovery

     -hide net.jini -hide com.sun.jini

     examples.sunw.Client                              [roots]
     net.jini.lookup.ServiceDiscoveryManager
                      $LookupCacheImpl$LookupListener_Stub
```

ClassDep – Client Classpath-B

```
java -cp <jini_home>/lib/tools.jar:<jdk_home>/lib/tools.jar
     com.sun.jini.tools.ClassDep
     -cp <jini_home>/classes:<sunw_home>/classes -files

     -in net.jini -in com.sun.jini -in examples.sunw

     -out net.jini.activation -out net.jini.config
     -out net.jini.constraint -out net.jini.core
     -out net.jini.export     -out net.jini.id
     -out net.jini.iiop       -out net.jini.io
     -out net.jini.jeri       -out net.jini.jrmp
     -out net.jini.loader     -out net.jini.security
     -out net.jini.url        -out com.sun.jini.discovery

     -hide examples.sunw

     examples.sunw.Client                              [roots]
     net.jini.lookup.ServiceDiscoveryManager
                      $LookupCacheImpl$LookupListener_Stub
```

Classpath-A – client.jar

- 'sunw' classes to client.jar
  - 'jini' classes hidden, jsk-platform excluded

```
examples/sunw/Client.class
examples/sunw/Client$1.class
examples/sunw/Client$CacheListener.class
examples/sunw/Client$LookupDiscoveryListener.class
examples/sunw/Service.class
```

Classpath-B – client.jar

- 'jini' namespace classes to client.jar
  - 'sunw' classes hidden, jsk-platform excluded

```
com/sun/jini/config/Config.class
com/sun/jini/proxy/BasicProxyTrustVerifier.class
com/sun/jini/proxy/ConstrainableProxyUtil.class
    ........
net/jini/discovery/DiscoveryManagement.class
net/jini/lease/LeaseRenewalManager.class
net/jini/lookup/LookupCache.class
net/jini/lookup/ServiceDiscoveryManager.class
net/jini/lookup/ServiceItemFilter.class
    ........
```

ClassDep – Client Codebase-A

```
java -cp <jini_home>/lib/tools.jar:<jdk_home>/lib/tools.jar
     com.sun.jini.tools.ClassDep
     -cp <jini_home>/classes:<sunw_home>/classes -files

     -in net.jini -in com.sun.jini -in examples.sunw

     -out net.jini.activation -out net.jini.config
     -out net.jini.constraint -out net.jini.core
     -out net.jini.export     -out net.jini.id
     -out net.jini.iiop       -out net.jini.io
     -out net.jini.jeri       -out net.jini.jrmp
     -out net.jini.loader     -out net.jini.security
     -out net.jini.url        -out com.sun.jini.discovery

     -hide net.jini -hide com.sun.jini

     com.sun.jini.proxy.BasicProxyTrustVerifier  [roots]
     net.jini.lookup.ServiceDiscoveryManager
                      $LookupCacheImpl$LookupListener_Stub
```

ClassDep – Client Codebase-B

```
java -cp <jini_home>/lib/tools.jar:<jdk_home>/lib/tools.jar
     com.sun.jini.tools.ClassDep
     -cp <jini_home>/classes:<sunw_home>/classes -files

     -in net.jini -in com.sun.jini -in examples.sunw

     -out net.jini.activation -out net.jini.config
     -out net.jini.constraint -out net.jini.core
     -out net.jini.export     -out net.jini.id
     -out net.jini.iiop       -out net.jini.io
     -out net.jini.jeri       -out net.jini.jrmp
     -out net.jini.loader     -out net.jini.security
     -out net.jini.url        -out com.sun.jini.discovery

     -hide examples.sunw

     com.sun.jini.proxy.BasicProxyTrustVerifier  [roots]
     net.jini.lookup.ServiceDiscoveryManager
                      $LookupCacheImpl$LookupListener_Stub
```

Codebase-A – client-dl.jar

- No 'sunw' classes to client-dl.jar
  - Client defines no 'downloadable' classes of its own
    - No proxies
    - No trust verifiers
  - Client provides downloadable 'jini' classes through its use of the SDM
    - Remote event listener
  - 'jini' classes hidden, jsk-platform excluded

Codebase-B – client-dl.jar

- 'jini' namespace classes to client-dl.jar
  - 'sunw' classes hidden, jsk-platform excluded
  - Only need listener stub when using JRMP
- Could simply use sdm-dl.jar in this case
  - Provided for convenience when client uses the ServiceDiscoveryManager

```
com/sun/jini/proxy/BasicProxyTrustVerifier.class
net/jini/lookup/ServiceDiscoveryManager
      $LookupCacheImpl$LookupListener_Stub.class
```

Packaging The Client

- client.jar

```
cd <sunw_home>/classes
jar cvf <lib>/client.jar <Classpath-A output>

cd <jini_home>/classes
jar uvf <lib>/client.jar <Classpath-B output>
```

- client-dl.jar

```
cd <jini_home>/classes
jar cvf <lib>/client-dl.jar -C <manifest>/client-dl/
   META-INF/PREFFERRED.LIST <Codebase-B output>
```

Configuration

- Service configuration
  - For ServiceStarter
    - <sunw>/<jeri|jrmp|jsse>/config/start-service.config
  - For Service
    - <sunw>/<jeri|jrmp|jsse>/config/service.config
    - Exporter – for the service
    - Proxy preparer(s) – discovery utilities & JoinManager
- Client configuration
  - <sunw>/<jeri|jrmp|jsse|>/config/client.config
  - Exporter – SDM event listener
  - Proxy preparer(s)
    - Lookup service proxy – discovery utilities
    - Service proxy – SDM/cache filter

ServiceStarter Configuration

```
import com.sun.jini.start.NonActivatableServiceDescriptor;
import com.sun.jini.start.ServiceDescriptor;

com.sun.jini.start {

  private static codebase  = "http://<host>:<port>/service-dl.jar";
  private static policy    = "<common>${/}local.policy";
  private static classpath = "<sunw>${/}lib${/}service.jar";
  private static implName  = "examples.sunw.RemoteServiceImpl";
  private static config    = "<sunw>${/}config${/}service.config";
  private static args      = new String[] { config };

  private sd = new NonActivatableServiceDescriptor
                   (codebase,policy,classpath,implName,args);

  static serviceDescriptor = new ServiceDiscriptor[] { sd };

}//end com.sun.jini.start
```

Service Configuration (JERI)

```
/* appropriate import statements */

examples.sunw.service {

  private static groupsToJoin = new String[] {"SUNW_Group0",
                                               "SUNW_Group1"};

  static discoveryManager = new LookupDiscoveryManager
                                      (groupsToJoin,
                                       new LookupLocator[] {},
                                       null,
                                       this);

  static serverExporter = new BasicJeriExporter
                                (TcpServerEndpoint.getInstance(0),
                                 new BasicILFactory(),
                                 false, true);
}//end examples.sunw.service
```

Client Configuration (JRMP)

```
/* appropriate import statements */

examples.sunw.client {

  private static groupsToDiscover =
                                  new String[] {“SUNW_Group0”,
                                                “SUNW_Group1”};
  private static locsToDiscover = new LookupLocator[]
       { new LookupLocator(ConfigUtil.getHostname(),4160) };

  static discoveryManager = new LookupDiscoveryManager
                                              (groupsToDiscover,
                                               locsToDiscover,
                                               null,
                                               this);
}//end examples.sunw.client

net.jini.lookup.ServiceDiscoveryManager {

  static eventListenerExporter = new JrmpExporter();

}//end net.jini.lookup.ServiceDiscoveryManager
```

Security Policy – Guideline

- Trust 'local' code – classpath
  - Code you installed
    - The JDK
    - The starter kit
    - The client or service you wrote/installed
  - Grant all permissions to specific classpaths
  - Never use policy.all (grants to all classpaths)
- Do not trust 'foreign' code – codebase
  - Proxy code that executes in your VM
  - Grant permissions only to code from sources whose trust can be verified
    - Trusted principals
    - Grant no more than what's necessary

JERI/JRMP Security Policy

- Grant all permissions to local classpath
- Do not grant permissions to codebase (*-dl.jar files)

```
grant codebase "file:<jini-lib>/jsk-platform.jar {
    permission java.security.AllPermission "", "";
};
grant codebase "file:<jini-lib>/start.jar {
    permission java.security.AllPermission "", "";
};
grant codebase "file:<sunw-lib>/sunw-service.jar {
    permission java.security.AllPermission "", "";
};
grant codebase "file:<sunw-lib>/sunw-client.jar {
    permission java.security.AllPermission "", "";
};
```

JERI, JRMP Command Lines

- Starting the service and client

```
<java-home>/bin/java -cp <jini-lib>/start.jar
  -Djava.security.manager=
  -Djava.security.policy=<common>/local.policy
  -Djava.security.policy=<common>/jini.logging
  [-Djava.rmi.server.hostname=<hostname>]
  [-Djava.security.debug=access,failure]
  com.sun.jini.start.ServiceStarter
                                <sunw>/sunw-service.config
  [optional config override values]


<java-home>/bin/java -cp <jini-lib>/jsk-platform.jar:
                         <sunw-lib>/sunw-client.jar
  -Djava.security.manager=
  -Djava.security.policy=<common>/local.policy
  -Djava.security.policy=<common>/jini.logging
  [-Djava.rmi.server.hostname=<hostname>]
  [-Djava.security.debug=access,failure]
  examples.sunw.Client <sunw>/sunw-client.config
  [optional config override values]
```

A Strategy For Debugging

- Start the Jini infrastructure
  - HTTP server
  - Lookup service (if activatable, start Phoenix)
  - Browser
  - Should be stable, no need to shutdown
- Start the service
  - Verify with Browser
  - Shutdown administratively with Browser
  - When satisfied, leave it up
- Start the client
  - Use the client to test the service
  - Client typically comes and goes

Deploying For Secure Remote Communication Using JERI/JSSE

Sun microsystems
We make the net work.

Network Security Requirements

- Provide basic network security for remote calls
  - Mutual authentication (client and server)
  - Authorization (access control)
  - Object Integrity (cryptographic checksums)
  - Confidentiality (encryption)
- Provide a security model that addresses downloaded code
- Integrate with the J2SE security model
- Configure network security at deployment time

Dealing With Downloaded Code

- Must be able to verify object integrity
  - Verify both data and downloaded code
- Must be able to verify proxy is trusted
  - In the presence of downloaded code
- Must provide mutual authorization
  - Server authorizes clients
  - Client authorizes the downloaded proxy code

The Programming Model

- Defines a constraint system to express network security (and other) needs
- Defines a proxy interface to set constraints (RemoteMethodControl)
- Ties authentication to JAAS Subjects
- Provides mechanisms and a model for
  - Verifying object integrity of downloaded code
  - Verifying trust in proxies
  - Dynamically granting permissions to proxies
- Provides Jini ERI to support this model

Secure Remote Call Model

- Obtain a proxy from somewhere
- Perform 'proxy preparation'
  - Verify that you can trust the proxy
  - Attach constraints to the proxy
  - Grant necessary permissions to the proxy
- Make remote calls through the proxy

Configuring For Network Security

- Exporters for remote objects
  - Service backend, remote listener backend, etc.
  - Set only integrity as an invocation constraint
  - For flexibility, base server-side access control on Java policy
    - Don't specify client principal constraints
- Preparers for received proxies
  - Service proxies, leases, listener proxies, etc.
  - Set server principal constraints
    - Integrity, client authentication, server authentication, ServerMinPrincipal

Server Authentication

- Saying YES is not enough
- Need to control who server authenticates as
- Use ServerMinPrincipal constraint to control who
  - As at least KerberosPrincipal “joe@realm”
  - As at least X500Principal “CN=Jack”
  - As at least GroupPrincipal “admin”

PKI/JSSE Concepts

- Keystore
  - Contains both private and public key info
  - Associated with a password
  - Never share with others
- Login configuration file
  - Points your system at private info
- Truststore
  - Keystore that contains only public key info
  - Share with others
    - Tells others who you are (principals)
    - Provides public keys for encrypted communication
    - Use certFile to share self-signed public key certificates
    - Certificate Authority (CA) more flexible than self-signed

ServiceStarter Configuration

```
import com.sun.jini.start.NonActivatableServiceDescriptor;
import com.sun.jini.start.ServiceDescriptor;
import net.jini.url.httpmd.HttpmdUtil;

com.sun.jini.start {

  private static codebase  = HttpmdUtil.computeDigestCodebase
                  ("<jini-lib>",
                   "httpmd://<host>:<port>/service-dl.jar;sha=0");
  private static policy    = "<sunw>${/}service.policy";
  private static classpath = "<sunw>${/}lib${/}service.jar";
  private static implName  = "examples.sunw.RemoteServiceImpl";
  private static config    = "<sunw>${/}config${/}service.config";
  private static args      = new String[] { config };

  private sd = new NonActivatableServiceDescriptor
                     (codebase,policy,classpath,implName,args);

  static serviceDescriptor = new ServiceDiscriptor[] { sd };

}//end com.sun.jini.start
```

Service Configuration (JSSE)

```
/* appropriate import statements */
shared.entries {
  1.) Retrieve principals from the public key truststore:
      -- service, lookup, browser, client --
  2.) Configure registrarPreparer for discovery utilities
}//end shared.entries

examples.sunw.service {
  1.) Configure serverExporter used to export the service
  2.) Configure loginContext (for Subject, private key info)
  3.) Service-specific items: discoveryManager
}//end examples.sunw.service

net.jini.discovery.LookupDiscovery {
  registrarPreparer = shared.entries.registrarPreparer;
}//end net.jini.discovery.LookupDiscovery

net.jini.discovery.LookupLocatorDiscovery {
  registrarPreparer = shared.entries.registrarPreparer;
}//end net.jini.discovery.LookupLocatorDiscovery

net.jini.lookup.JoinManager {
  1.) Configure preparers: registrar, registration, serviceLease
}//end net.jini.lookup.JoinManager
```

Shared Entries

```
shared.entries {

  /* Retrieve all required principals from truststore */

  private registrarConstraints =
   new BasicMethodConstraints(new InvocationConstraints
            ( new InvocationConstraint[]
                     { Integrity.YES,
                       ClientAuthentication.YES,
                       ServerAuthentication.YES,
                       new ServerMinPrincipal(reggiePrincipal)
                     },
              null)//preferences
   );
  private dynamicPermissions = new Permission[]
            { new AuthenticationPermission(servicePrincipal,
                                           reggiePrincipal,
                                           "connect") };
  registrarPreparer = new BasicProxyPreparer
                                    (true, //verify trust
                                     registrarConstraints,
                                     dynamicPermissions);

}//end shared.entries
```

Exporter And Login

```
examples.sunw.service {
  private serviceEndpoint = SslServerEndpoint.getInstance(0);
  private serviceConstraints =
   new BasicMethodConstraints
          (new InvocationConstraints
                (new InvocationConstraint[]{Integrity.YES},
                 null)//preferences
          );

  //RemoteServicePermission is for policy-based access control
  private serviceILFactory = new ProxyTrustILFactory
                              (serviceConstraints,
                               RemoteServicePermission.class);

  serverExporter = new BasicJeriExporter(serviceEndpoint,
                                         serviceILFactory);

  //Set property: -Djava.security.auth.login.config
  loginContext = new LoginContext("sunw-service.jaas.login");

  static discoveryManager = LookupDiscoveryManager(...);

}//end examples.sunw.service
```

For JoinManager

```
net.jini.lookup.JoinManager {

  /* No need for verification, constraints, or permissions */
  registrarPreparer = new BasicProxyPreparer();

  /* Secondary proxies only need constraints set */
  registrationPreparer = new BasicProxyPreparer
                         (false,
                          shared.entries.registrarConstraints,
                          null);

  serviceLeasePreparer = new BasicProxyPreparer
                         (false,
                          shared.entries.registrarConstraints,
                          null);

}//end net.jini.lookup.JoinManager
```

Client Configuration (JSSE)

```
/* appropriate import statements */

shared.entries {
  /* similar to shared entries for service configuration */
}//end shared.entries

examples.sunw.client {

  1.) Configure servicePreparer
  2.) Configure loginContext (for Subject, private key info)
  3.) Service-specific items: discoveryManager

}//end examples.sunw.service

/* discovery utility blocks same as for service configuration */

net.jini.lookup.ServiceDiscoveryManager {

  1.) Configure exporter: eventListener
  2.) Configure preparers: registrar, eventLease

}//end net.jini.lookup.ServiceDiscoveryManager
```

Service Preparer And Login

```
examples.sunw.client {

  private serviceProxyConstraints =
   new BasicMethodConstraints(new InvocationConstraints
    (new InvocationConstraint[]
      { Integrity.YES,
        ClientAuthentication.YES,
        ServerAuthentication.YES,
        new ServerMinPrincipal(shared.entries.servicePrincipal)
      },
      null)//preferences
   );
  private dynamicPermissions = new Permission[]
  {new AuthenticationPermission(shared.entries.clientPrincipal,
                                shared.entries.servicePrincipal,
                                "connect")};
  servicePreparer = new BasicProxyPreparer
                                   (true, //verify trust
                                    serviceProxyConstraints,
                                    dynamicPermissions);
  loginContext = new LoginContext("sunw-client.jaas.login");
  static discoveryManager = LookupDiscoveryManager(...);

}//end examples.sunw.client
```

For ServiceDiscoveryManager

```
net.jini.lookup.ServiceDiscoveryManager {

  private listenerEndpoint = SslServerEndpoint.getInstance(0);
  private listenerConstraints =
      new BasicMethodConstraints
            (new InvocationConstraints
                  (new InvocationConstraint[]{Integrity.YES},
                   null)//preferences
            );
  private serviceILFactory = new ProxyTrustILFactory
                                      (listenerConstraints,
                                       AccessPermission.class);

  eventListenerExporter = new BasicJeriExporter
                                            (serviceEndpoint,
                                             serviceILFactory);
  registrarPreparer = new BasicProxyPreparer();
  eventLeasePreparer =
     new BasicProxyPreparer
            (false, shared.entries.registrarConstraints, null);

}//end net.jini.lookup.ServiceDiscoveryManager
```

CheckConfigurationFile Tool

- Checks your configuration for errors
  - Provide an entry description file for each block

```
<java-home>/bin/java -cp <jini-lib>/tools.jar
  com.sun.jini.tool.CheckConfigurationFile
  -cp <jini-lib>/jsk-platform.jar:<jini-lib>/jini-ext.jar:
      <sunw-lib>/sunw-service.jar
  -entries <jini-home>/configentry/LookupDiscovery:
           <jini-home>/configentry/LookupLocatorDiscovery:
           <jini-home>/configentry/JoinManager:
           <sunw>/configentry/shared-entries:
           <sunw>/configentry/sunw-service
  <sunw>/sunw-service.config

<java-home>/bin/java -cp <jini-lib>/tools.jar
  com.sun.jini.tool.CheckConfigurationFile
  -cp <jini-lib>/jsk-platform.jar:<jini-lib>/jini-ext.jar:
      <sunw-lib>/sunw-client.jar
  -entries <jini-home>/configentry/LookupDiscovery:
           <jini-home>/configentry/LookupLocatorDiscovery:
           <jini-home>/configentry/ServiceDiscoveryManager:
           <sunw>/configentry/shared-entries:
           <sunw>/configentry/sunw-client
  <sunw>/sunw-client.config
```

Security Policy – Guideline

- Trust 'local' code – classpath
  - Code you installed
  - Grant all permissions to specific classpaths
  - Never use policy.all (grants to all classpaths)
- Do not trust 'foreign' code – codebase
  - Grant permissions only to code from sources whose trust can be verified
    - Trusted principals
- Server side access control based on Java policy, not client principal constraints

Service Security Policy (JSSE)

```
keystore "file:<common>/sunw.truststore;

grant codebase "file:<jini-lib>/jsk-platform.jar {
    permission java.security.AllPermission "", "";
};
grant codebase "file:<sunw-lib>/sunw-service.jar {
    permission java.security.AllPermission "", "";
};
grant principal "sunw-client" {
    permission examples.sunw.RemoteServicePermission
                                    "getProxyVerifier";
    permission examples.sunw.RemoteServicePermission
                                    "getPrice";
};
grant principal "sunw-browser" {
    permission examples.sunw.RemoteServicePermission
                                    "getProxyVerifier";
    permission examples.sunw.RemoteServicePermission
                                    "getAdmin";
    ..............
    permission examples.sunw.RemoteServicePermission
                                    "destroy";
};
```

Client Security Policy (JSSE)

```
keystore "file:<common>/sunw.truststore;

grant codebase "file:<jini-lib>/jsk-platform.jar {
    permission java.security.AllPermission "", "";
};

grant codebase "file:<sunw-lib>/sunw-client.jar {
    permission java.security.AllPermission "", "";
};

grant principal "sunw-reggie" {
    permission net.jini.security.AccessPermission "notify";

};
```

Service Command Line (JSSE)

```
<java-home>/bin/java -cp <jini-lib>/start.jar
  -Djava.security.manager=
  -Djava.security.policy=<common>/local.policy

  -Djava.security.auth.login.config=<sunw>/service.login
  -Djavax.net.ssl.trustStore=<common>/sunw.truststore
  -Djava.protocol.handler.pkgs=net.jini.url
  -Djava.security.properties=
                    <common>/dynamic-policy.properties

  -Djava.security.policy=<common>/jini.logging
  [-Djava.rmi.server.hostname=<hostname>]
  [-Djava.security.debug=access,failure]
  com.sun.jini.start.ServiceStarter
                     <sunw>/sunw-service.config
                     [optional config override values]
```

Client Command Line (JSSE)

```
<java-home>/bin/java -cp <jini-lib>/jsk-platform.jar:
                         <sunw-lib>/sunw-client.jar
  -Djava.security.manager=
  -Djava.security.policy=<sunw>/client.policy

  -Djava.security.auth.login.config=<sunw>/client.login
  -Djavax.net.ssl.trustStore=<common>/sunw.truststore
  -Djava.protocol.handler.pkgs=net.jini.url
  -Djava.security.properties=
                    <common>/dynamic-policy.properties

  -Djava.security.policy=<common>/jini.logging
  [-Djava.rmi.server.hostname=<hostname>]
  [-Djava.security.debug=access,failure]
  examples.sunw.Client <sunw>/sunw-client.config
                     [optional config override values]
```

Some Random Debugging Tips

- Set net.jini.jeri.level = FAILED – FINE
  - 'Dynamic grants not supported'
    - Extenstions directory may not contain jsk-policy.jar
    - May be missing policy provider
    - Look for typo in -Djava.security.properties value
  - 'access denied (AccessPermission notify)'
    - May not be logging in (correctly)
    - Verify loginContext is configured correctly
    - Look for problems with config file name specification
  - 'unsupported constraint' - (null Subject)
    - Code executed outside of Subject.doAsPrivileged?
- Use DebugDynamicPolicyProvider tool

DebugDynamicPolicyProvider

- Convenient mechanism for determining necessary permissions (..grantAll = true)
  - Can produce a log of all required permissions
    - Doesn't deny required permission and fail
    - Logs the permission in correct format and continues
    - Uses java.util.logging.Logger for (less verbose) output
  - Use as is or modify output for portability
    - X.509 certificate principal names to KeyStore aliases
    - System property substitutions
- Debug current policy (..grantAll = false)
  - Logs info about thrown security exceptions

Debug Policy Strategies

- Debug/refine existing policy
  - Set ..grantAll = false
    - Stops on first unhandled 'access denied' encounter
  - Set logging level = FINE or FINER
    - Logs missing permissions with stack trace for context
    - Output more useful than old trial-and-error method
- Populate 'empty' policy
  - Set ..grantAll = true
    - Doesn't stop, logs missing permissions and continues
  - Set net.jini.security.policy.level = WARNING/FINE
    - WARNING gives cleaner output, but stack trace from FINE might tell you 'why' permission is needed
  - Get first cut at policy, then use ..grantAll=false

Debug Policy Usage

- Get JAR from starterkit-examples project
  - <install>/src/com/sun/jini/tool/lib20/debugpolicy.jar
  - Copy debugpolicy.jar to <jdk>/jre/lib/ext directory
- Create policy provider properties file
  - <common>/debug-policy.properties

```
policy.provider=
    com.sun.jini.tool.DebugDynamicPolicyProvider
com.sun.jini.tool.DebugDynamicPolicyProvider.
                        grantAll=true (or false)
```

- Set logging level to WARNING or FINE
  - **net.jini.security.policy.level**
- Tell the VM what policy provider to use

```
-Djava.security.properties=
                    <common>/debug-policy.properties
```

Example Command Line

```
<java-home>/bin/java -cp <jini-lib>/jsk-platform.jar:
                         <sunw-lib>/sunw-client.jar
  -Djava.security.manager=
  -Djava.security.policy=<sunw>/client.policy

  -Djava.security.auth.login.config=<sunw>/client.login
  -Djavax.net.ssl.trustStore=<common>/sunw.truststore
  -Djava.protocol.handler.pkgs=net.jini.url
  -Djava.security.properties=
                    <common>/debug-policy.properties

  -Djava.security.policy=<common>/jini.logging
  [-Djava.rmi.server.hostname=<hostname>]
  [-Djava.security.debug=access,failure]
  examples.sunw.Client <sunw>/sunw-client.config
                       [optional config override values]
```

Wrap Up

Sun microsystems
We make the net work.

Summary – Some Useful Tips

- Think of development and deployment as separate activities
  - Many learning curve issues are related to deployment, not development
    - The line often blurs in places
- Develop with deployment in mind
  - Exploit runtime configuration
    - Pluggable protocols (JRMP, JERI, HTTP, JSSE, etc.)
    - Don't hard code (especially exporters or preparers)
  - Always ready for security
    - Login context
    - Exporters
    - Proxy preparers

More Tips

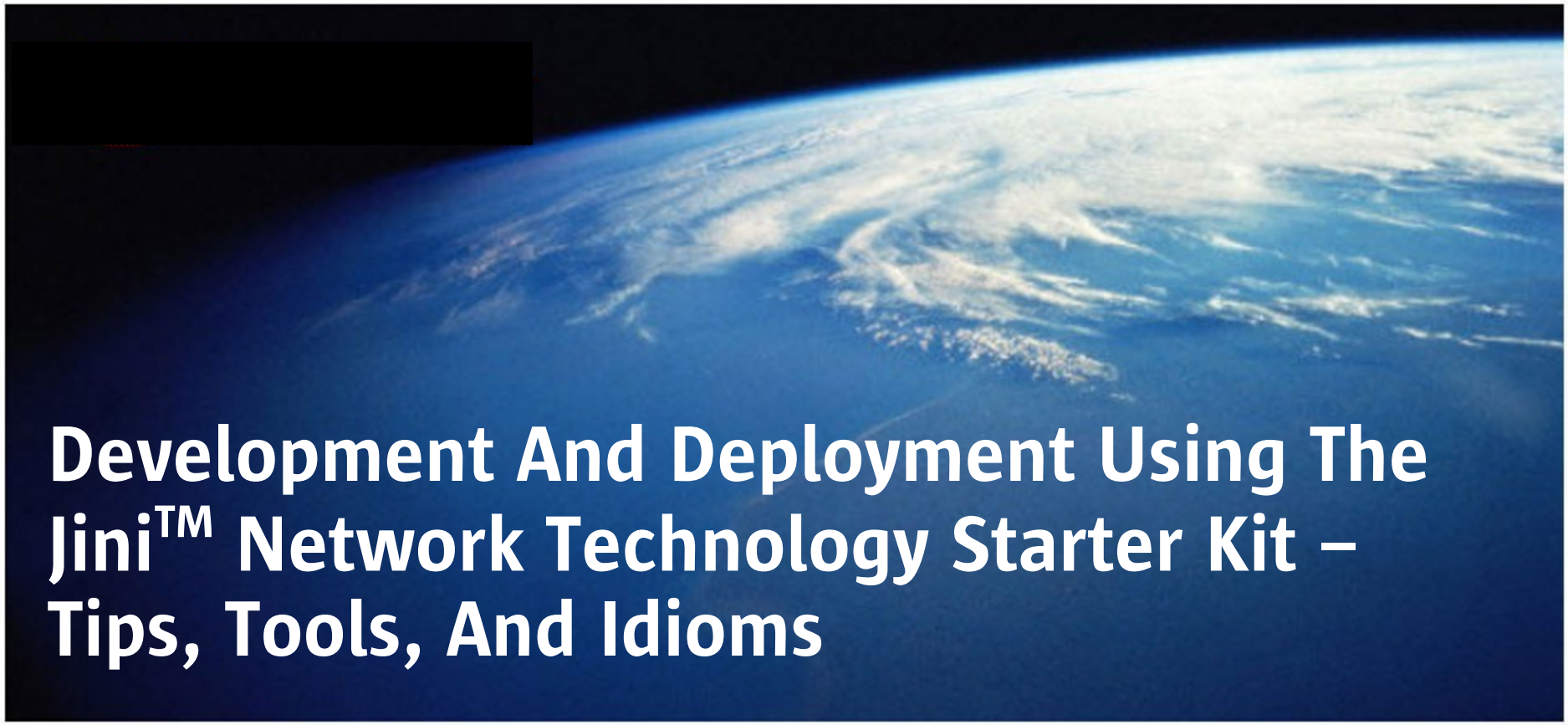
- Build for deployment
  - Preferred classes
  - ClassDep tool
  - Packaging: <app>.jar/<app>-dl.jar
- A debug strategy to consider
  - Start Jini infrastructure and leave it up
    - HTTPD
    - Lookup service (Phoenix if activatable)
    - Browser for visibility and administrative shutdown
  - Service generally long lived
    - Make administratable for graceful shutdown
  - Client usually less long lived (at least for debug)
    - User controlled (GUI, prompt user, etc.)

Still More Tips

- Explicitly export your remote objects
  - Secure configurations: base server-side access control on Java policy not client principal constraints
- Always prepare received proxies
  - Helper utilities provide much relief
  - For SDM/cache: pass proxy preparer as filter
  - Secure configurations: set server principal constraints – integrity, client authentication, server authentication, ServerMinPrincipal
- Useful debug tools
  - CheckConfigurationFile
  - DebugDynamicPolicyProvider

References

- Starter kit manpages
  - <jini>/doc/api/com/sun/jini/tool
    - ClassDep.html
    - CheckConfigurationFile.html
  - <jini>/doc/api/com/sun/jini/start
    - package-summary.html (ServiceStarter)
- Policy file syntax
  - http://java.sun.com/j2se/1.4./docs/guide /security/PolicyFiles.html
- Hello world example in the starter kit
- Example projects on www.jini.org
  - starterkit-examples
  - user-btmurphy



Development And Deployment Using The Jini™ Network Technology Starter Kit – Tips, Tools, And Idioms

Brian Murphy

brian.t.murphy@sun.com

Sun microsystems
We make the net work.