The Jtrix dictionary

$Id: jtrix-dictionary.lyx,v 1.8 2001/11/21 10:30:04 nik Exp $

Nik Silver
feedback@jtrix.org
Jtrix Ltd
57–59 Neal Street, London WC2H 9PJ, UK
+44 20 7395 4990

Contents

1 Useful phrases

1.1 Access point netlet

A netlet which is downloaded and run in response to a service binding from a client.

Such a netlet tends to act as a gateway to other netlets which perform the bulk of the work. However, this is a loose description since any netlet can include any amount of intelligence and may range from being a simple proxy to a complex monolithic application netlet.

1.2 Beatrix

A framework for Jtrix applications.

Netlets need to worry about a lot of things: providing facets and services, deploying other netlets, finding resources, keeping in touch with other netlets in the application, and so on. Beatrix provides a number of ready-made netlets which can be extended and co-ordinated by means of plugins.

Beatrix defines a particular pattern for an application: it should have manager netlets, which keep the application alive, worker netlets, which use resources and do the bulk of the work, and access points netlets, which are downloaded in response to service requests. See Figure 1. Applications are not obliged to use Beatrix.

1.3 Bind server

A network resource enabling a node to bind a service.

A bind server is usually referenced by just a URL and will handle some kind of binding protocol. A bind server is usually part of a SAS.

1.4 Binding

Broadly, when one thing attaches to another.

See: facet binding, netlet binding, resource binding, service binding, warrant binding.

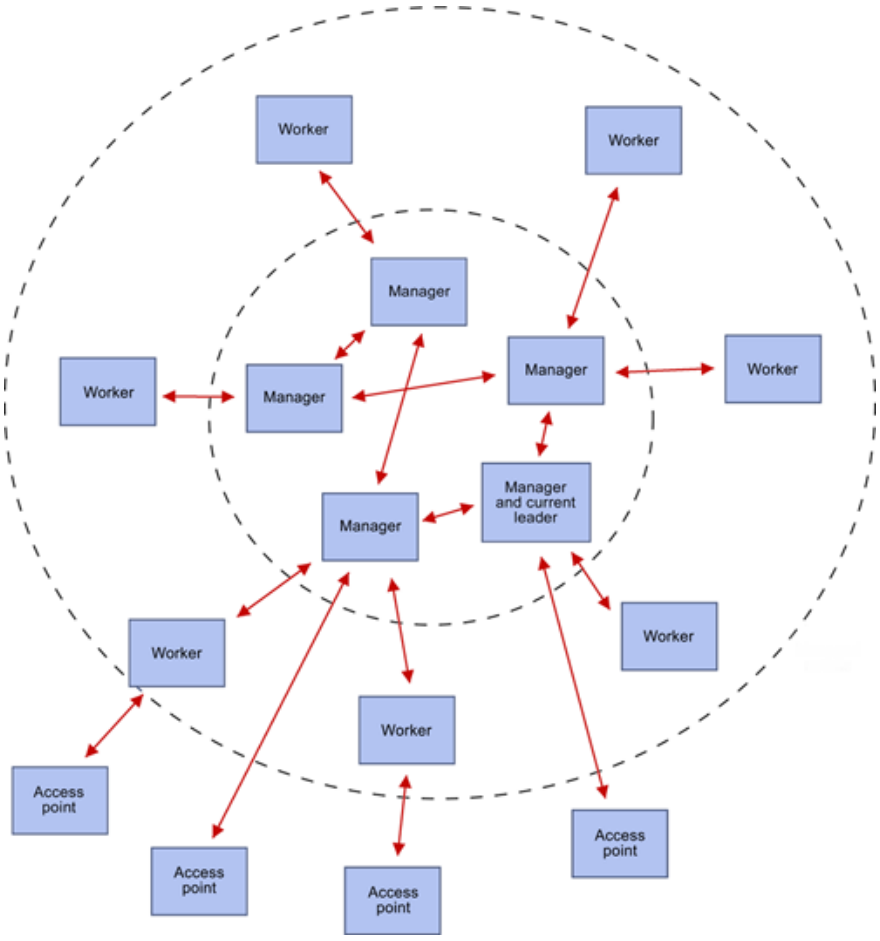

Figure 1: The pattern of a Beatrix application.

1.5 Binding protocol

A protocol used to fetch a netlet descriptor and unsigned argument.

A SAS uses binding protocols. The simplest binding protocol is HTTP, but other options are possible, for example to allow negotiation of an appropriate netlet descriptor.

A binding protocol is likely to be platform independent, not tied to any one programming language.

1.6 Bootstrap netlet

A netlet which is given to Jnode on the command line for it to run.

Jnode runs each bootstrap netlet in turn, then continues running so the netlets can continue their work. Jnode also endows each bootstrap netlet with the ability to access file I/O streams, also specified on its command line. This enables them to perform their own specific tasks (e.g. reading a warrant or writing a descriptor) without having the complete system access that an external netlet is given.

1.7 Contract

An agreement between a service provider and a consumer.

A contract is not formally represented within Jtrix—it is something which may well be arranged off-line or by some on-line means other than Jtrix.

However, an application may well keep a database of contracts and facilities available to the contractees. Then when a warrant is issued to the consumer it can contain a reference to the contract, and details can be updated as they use the service.

1.8 Descriptor

Usually shorthand for “netlet descriptor”, though other kinds of descriptor do exist.

1.9 External netlet

The program which booted a node and is seen by the node as a specially privileged netlet.

A node does not have any user interface, and it does not have any direct access to the outside world. This would make it rather difficult to control, and even to boot, hence the external netlet. An external netlet is therefore any ordinary application which happens also to be a netlet— it can boot the node and has other administration control over it. Since the external netlet is an ordinary program it has the usual access to system resources, unhindered by the node's security.

External netlets are how a Jtrix node can be embedded in another application.

1.10 Facet

An interface to access a feature in a service.

Given a service session a netlet can bind a facet. Then it can invoke methods on this facet to use the requested service.

For example, a mailing list service may offer one facet to manage the list and one facet to access the consumer's account state. All facets implement the `IRemote` interface.

As well as netlets offering facets, nodes may offer facets also. Node facets are not part of a service session; permission to use them is implicit in the netlet being allowed to run on that node—i.e. it is part of the hosting contract.

1.11 Facet binding

The act of a netlet attaching to a facet.

See: the `IRemote` interface, the `IService` interface.

1.12 Hospitality

The name of Jtrix.org's hosting service.

It is designed to sit on top of Jnode.

1.13 Hosting service

A service which allows a netlet to launch itself onto a node (probably from a remote location), run there, and access system resources.

There are typically three modes of access to a hosting service:

- The service owner (i.e. node owner) can grant warrants to its clients.
- The hosting client has this warrant which enables them to run netlets and use resources on the owner's node(s). This will typically be used by their management netlets: these will oversee smooth running of the application, start up small worker netlets and allocate them system resources.
- The worker netlets will use the system resources granted to them by their managers. But it is not their job to start new netlets or organise further resources, so they would not have access to these features of the hosting service.

1.14 Jnode

The Nodality node with facilities to make it executable from a command line, and with group communication and resource management.

Jnode acts as the external netlet for Nodality. As an external netlet all it does is run a series of "bootstrap netlets" specified on the command line, and then continues managing them. One example of a bootstrap netlet for Jnode is Jtrix.org's hosting service, called Hospitality. It uses the group communication facilities to keep various copies of itself working together in a distributed way.

With Jnode it is easy to run a netlet on a local machine, because it's just a case of running Jnode and naming the netlet descriptor.

1.15 Mediator

The part of a node that proxies connections from netlets to everything else, including system resources and facets.

The mediator ensures local security requirements are enforced. It is usually invisible to the netlets.

1.16 Netlet

A self-contained Jtrix application component.

A Jtrix application is made up of one or more netlets which communicate with each other. When a service is bound an access point netlet is delivered to fulfil that service.

See also: access point netlet, bootstrap netlet, external netlet, netlet binding, the `INetlet` interface.

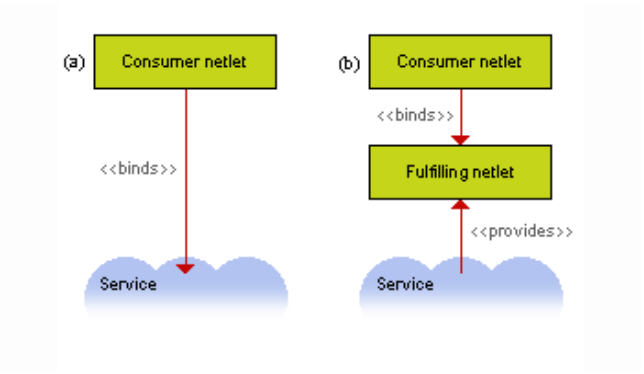

Figure 2: (a) A netlet binds to a service. But the reality is that (b) the service is fulfilled by another netlet, an access point netlet.

1.17 Netlet binding

Another way of expressing service binding, but emphasising that a netlet (an access point netlet) is delivered in order to fulfil the service.

See Figure 2.

1.18 Netlet descriptor

XML that tells a node enough for it to bind a netlet.

Among other things it must tell the node where to get the code from, so it will contain either URLs for the netlet code, or the code itself, or a combination. See Figure 3.

This code, or its URL, allows the node to load the netlet. In binding it, the netlet's initialisation method will be called, but this may require some parameters. So a descriptor also contains a parameter object which it passes to the initialisation method.

Thus a descriptor contains code (or a URL for the code) plus an initialisation parameter, and this is enough to load and bind and a netlet.

The contents of a descriptor is signed to prevent tampering.

See also: signed argument, unsigned argument.

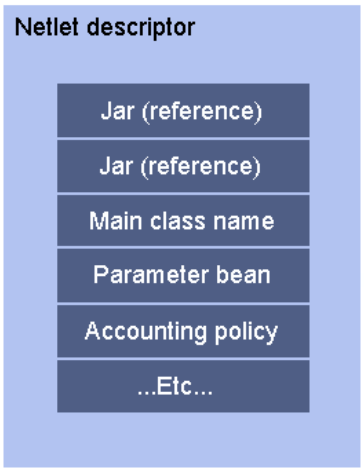

Figure 3: A netlet descriptor contains enough information to load and run a netlet. That includes the codebase (or references to it) and an initialisation object among other things.

1.19 Nodality

Jtrix.org's implementation of a Jtrix node.

Nodality is designed to span only one machine, but several Nodality instances may run on that machine simultaneously.

See also: Jnode.

1.20 Node

Virtual environment in which a Jtrix netlet runs and which manages access to resources and services.

We can imagine that this environment is based in a single server or PC. But this need not be the case—it could physically exist over several machines working together.

A node must conform to a particular API but may have many implementations.

See also: Nodality, the `INode` interface.

1.21 Resource

A node-specific facility such as disk space or networking sockets.

Resources need to be explicitly requested and bound, and each is tied to a specific node. Each resource will only be available to one netlet at a time, but may be divided into small chunks: a 2GB disk may be may divided into two thousand 1MB disk resources; a single IP address may be divided into several socket factories, each of which makes no more than 16 sockets available.

Most netlets need resources, but this is by no means necessary. For example, the node may well provide a message bus via one of its own facets, thus allowing netlets to communicate with each other without formally requesting resources. This is how Beatrix's manager netlets co-ordinate their workers. A node's message bus isn't the same as a network socket resource because a netlet's access to the message bus operates at a very high level.

In general resources can be anything, and might include access to data on a CD-ROM, triply-redundant disk space, load balanced IP addresses, and so on.

See also: resource binding.

1.22 Resource binding

The process of a netlet attaching to a node's resources.

Unlike service binding, resource binding does not require a warrant because the very fact that the netlet is running on the node means it is there under an agreed contract and that contract also governs resource access.

1.23 SAS

A service advertisement service—a service which allows netlets to be downloaded and hence allows service binding.

More correctly, it allows the downloading of netlet descriptors and corresponding unsigned arguments. It may use the HTTP protocol, but other options are possible which allow, for example, negotiation for the most suitable netlet.

Since any netlet descriptor will probably contain references to the JARs it needs (rather than containing the JARs themselves) a SAS will most likely also deliver JAR files.

Access to a SAS is likely to be in three ways:

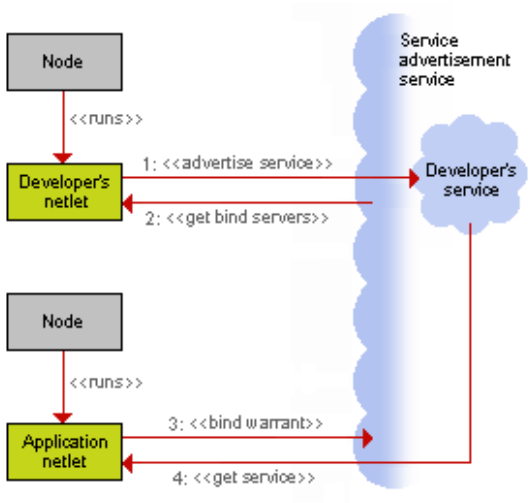

Figure 4: Using a service advertisement service (SAS). (1) A developer uploads their service to the SAS. (2) In return they get the URLs of bind servers which they put into warrants for their service. (3) & (4) A consumer netlet uses such a warrant to bind to the developer's service.

- The owner of the SAS will have administration access to it. They will be able to create contracts with customers and issue them with user warrants.
- A SAS user will use their warrant to upload netlet descriptors, unsigned arguments and JAR files. With each upload they will get in return a list of bind servers (i.e. URLs) where the uploaded items are now available. They can then incorporate these bind servers into warrants for their newly-uploaded service and pass on these warrants to their service consumers.
- A service consumer will present a warrant to its node which will then bind the service from the SAS. It will use the SAS because the warrant lists the bind servers where the service is available.

See Figure 4 for the second and third of these.

See also: binding protocol.

1.24 Service

Any on-line activity performed for others and which is generally available.

By "generally available" we mean that the business in general is the service, not a single transaction or a series of transactions taking place under a single agreement, and it should be available at any time from any location.

A service includes activities such as Web-based mail, credit card authorisation or news headlines. But a service can also include access to resources such as disk space, memory, CPU time, bandwidth and IP addresses.

1.25 Service binding

The process of a netlet requesting and getting access to a service.

Several steps are involved:

1. The client netlet presents to the node a warrant for the service.
2. The node examines the warrant and determines how to get a descriptor for the service. It may download it, it may already have a copy, or it may actually be embedded in the warrant.
3. The node creates the service netlet from the descriptor.

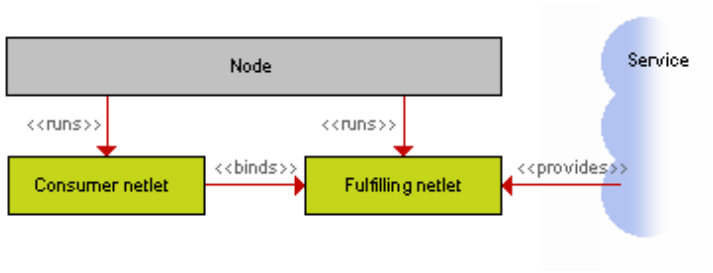

Figure 5: A client netlet binds to a service through another netlet. The netlet-to-netlet binding creates a service session. Both netlets need to sit on the same node. The service is fufilled through the netlet, and is more than just the netlet itself.

4. The node initialises the netlet.
5. The node creates a service session between the client netlet and the service netlet.

See Figure 5.

See also: binding protocol, facet, SAS.

1.26 Service connection

Another term for “service session”.

1.27 Service session

A continuous connection between two netlets allowing one to make use of a service from the other.

A service session is what a netlet gets in return for binding a warrant, and is the `IService` interface. From an `IService` interface a netlet can bind and use facets.

Although the service netlet provides an `IService` interface to the consumer netlet the arrangement is symmetrical: the service netlet also gets an `IService` interface from the consumer. Through this reciprocating consumer service interface the service netlet can bind facets from the consumer. For example, before the service netlet provides its service interface (and hence access to its service) it may expect access

to the consumer's wallet to extract payment. This would be achieved by it binding facets from the consumer's service interface.

1.28 Signed argument

An arbitrary argument object passed to a netlet on initialisation.

Also known as the "parameter bean", the signed argument is embedded in a netlet descriptor and hence is signed. It is passed to the netlet via the init method of the `INetlet` interface.

1.29 Unsigned argument

An arbitrary argument object passed to a netlet on initialisation.

Unlike the signed argument, the unsigned argument is not embedded in the netlet descriptor and instead is delivered alongside it as part of the binding protocol which delivers the descriptor. It is passed to the netlet via the init method of the `INetlet` interface.

Unsigned arguments exist because it is often convenient to generate and send quickly-changing data without having to sign it, which is computationally intensive. Also, while the netlet descriptor will be platform dependent (tied to a specific programming language), the unsigned argument, like the binding protocol, is not. Hence the unsigned argument is assumed to be no more than a stream of bytes.

1.30 Warrant

A right to use a service, represented as XML.

Another way to think of a warrant is that it is evidence of a contract. If a contract has been arranged (on- or off-line) then a warrant will result so that the service can be used and the contract fulfilled.

When a client netlet wants to use a service it presents its warrant to the node which then finds and binds the server netlet. Of course, for a node to use a warrant to find the right netlet the warrant needs to contain either a descriptor or a URL reference of where to find that descriptor—Figure 6. So to make use of a service, a netlet needs only a warrant. This contains everything a node needs to find and bind a service, however indirectly.

A warrant is not just a general right to use a service—it is very likely to be limited for the terms of the particular contract for which it was issued. Two netlets may have warrants to use a service, but one may

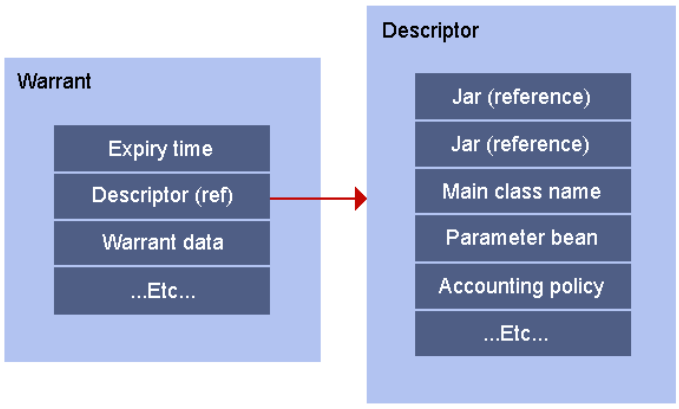

Figure 6: A warrant is a right to use a service. Among other things it must contain a descriptor for the relevant netlet, or a reference to one.

be time-limited, and another may be resource limited. That's up to whoever arranged the contract. In Figure 7 different warrants for the same service allow their holders access only to their area of that service.

The contents of a warrant is signed to prevent tampering.

1.31 Warrant binding

Another way of expressing service binding, but emphasising that a warrant is used.

2 Useful interfaces

2.1 IFacetCollection

Interface to a collection of facets. Nodes, netlets and services all offer facets, so their respective interfaces all implement this.

```
public interface IFacetCollection extends IFacetProvider
{
    /** Fetch names of available facets.
     * @return  The names of all the facets in this collection.
     */
    public String[] getFacets();
}
```

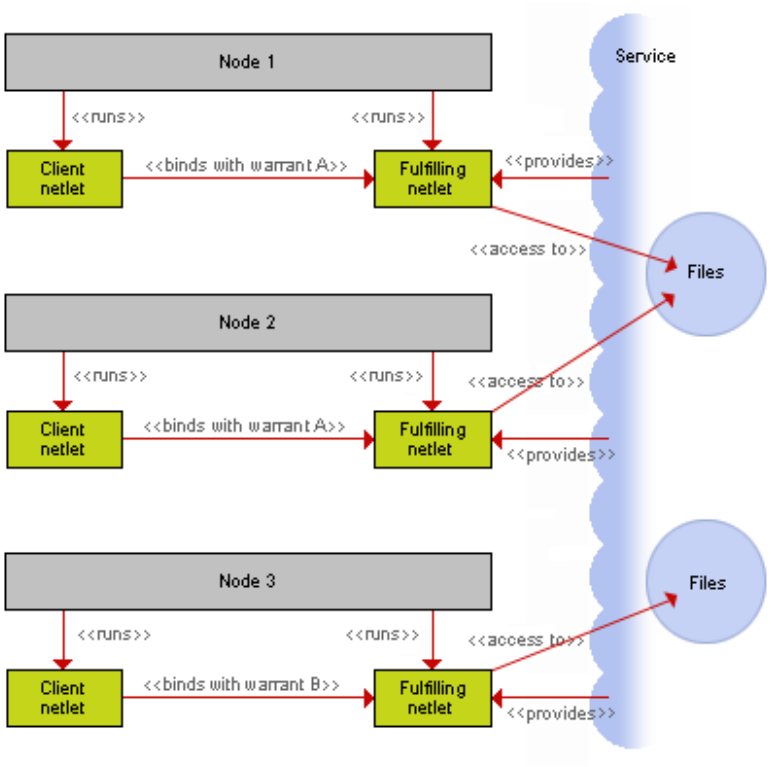

Figure 7: Warrants A and B are both for use of a file storage service. But warrant A limits its holder(s) to a specific group of files, while warrant B is limited to a different group. Thus each warrant-holder has access only to their files.

See also: the `IService` interface.

2.2 IFacetProvider

Interface for anything which provides facets, such as a service.

```
public interface IFacetProvider extends IRemote
{
    /** Bind to a facet by name
     * @param facet  The name of the facet. This should be the class name of
     *     the facet and the interface should be in the consumer's class space.
     * @throws FacetBindException  If an error occured while binding to the
     *     facet.
     * @return  The remote interface. This can then be cast into the interface
     *     of the required facet.
     */
    public IRemote bindFacet(String facet) throws FacetBindException;
}
```

See also: the `IService` interface.

2.3 INetlet

Interface which a class implements to be a netlet.

`INetlet` allows initialisation, service binding, and termination. Initialisation gives the netlet an interface to its node (`INode`), and both the signed and unsigned arguments from its descriptor.

The `INetlet` interface is presented by a netlet to a node. When another netlet accesses the netlet to use a service it gets an `IService` interface, not an `INetlet` interface.

```
public interface INetlet extends IFacetCollection
{
    /** Allows the node to initialise the netlet. Called at most once.
     *
     * @param node  Interface to the netlet's view of the node.
     * @param parameter_bean   Bean object for parameters from the descriptor.
     *     Since this comes from the descriptor, if the descriptor is signed
     *     then we can assume this data is equally trusted.
     * @param unsigned_arg  Arbituary data that arrived with the descriptor,
     *     but external to it. Hence it is not signed (unlike the
     *     <tt>parameter_bean</tt>) and it is therefore untrusted.
     * @throws InitialiseException  In case of failure an exception is thrown;
     *     the netlet can then expect to be terminated.
     * @return Some arbitrary data; it is determined by the implementation and
     *     the context; it must be properly intepreted by whoever executed
     *     this netlet. Null is a valid, non-error return.
     */
    public byte[] initialise(INode node,
                             Object parameter_bean,
                             byte[] unsigned_arg) throws InitialiseException;

    /** Allows the node to request termination of netlet.  Called before
     * netlet is actually shutdown regardless of initialisation state. May be
```

```
     * aborted without notice if takes too long. No threads should be left
     * when this function returns, otherwise they will be shut down, too.
     *
     * @param time  The time when the node will force a terminate even if the
     *     netlet has not yet terminated. This is seconds since 1 Jan 1970
     *     GMT.  When negative there is no fixed time, but the node may then
     *     terminate at any time. If this number is 0 then the node is
     *     informing the netlet that connections may be terminated
     *     asynchronously.
     * @param progress  The interface the netlet uses to notify the node of
     *     the progress of the termination. If the time is 0 then there is no
     *     reason to notify the node of progress.
     */
    public void terminate(long time, IShutdownProgress progress);

    /** Requests a service connection from this netlet.
     *
     * @param warrant   The warrant that was used to bind to this netlet.
     *     The implementation can assume that the node has checked the warrant
     *     by the time the implementation is called.
     * @param consumer  The consumer side of the connection. It is through
     *     this that the calling netlet offers its side of the two-way service
     *     connection.
     * @return  This side of the service connection. A null return is
     *     interpreted as a declined bind.
     */
    public IService bindService(Warrant warrant, IService consumer)
        throws ServiceBindException;

    /** Callback interface for the progress of a shutdown. This interface
     * will be given to a netlet by the node when it is requested to terminate.
     * The netlet can then use it inform the node of its shutdown progress.
     * Even though a node gives us a certain amount of time to shutdown
     * if we're seen to progressing well and we seem to need some extra time
     * then the node's administrator might be benevolent and give us a little
     * leeway.
     */
    public interface IShutdownProgress extends IRemote
    {
        /** Tell the node how we're progressing with our own shutdown.
         * @param progress  A percentage of the progress of the shutdown.
         */
        public void performed(double progress);
    }
}
```

2.4 INode

Interface through which a netlet talks to its node.

Primarily a netlet uses this to bind services using a warrant. But it is also used if the netlet wants to bind to node facets.

```
public interface INode extends IFacetCollection
{
    /** Request that the client netlet be terminated.
     */
    public void requestTermination();

    /** Ask the node to make a service connection. The node will instantiate a
     * service netlet. That netlet could be a proxy to the actual service.
     * @param warrant  The warrant that is used to bind to the service.
     * @param consumer  The consumer side of service. I.e. a service connection
```

```
     *     from the actual netlet making the request.
     * @return  The server side of the service.
     */
    public IService bindService(Warrant warrant, IService consumer)
        throws ServiceBindException;

    /** Create a service proxy, i.e. something with the same semantics
     * as a service connection but without binding a warrant.  This
     * proxy can be freely passed to other netlets.  Any facets passed
     * through the proxy will be dependent on the proxy, and will be
     * shutdown along with this proxy.  The proxy may have its own
     * thread pool.
     * @param impl  Something implementing the proxy
     * @param facet  The name of the interface that the proxy represents.
     *     The proxy object can then be cast into this.
     * @return  An interface to manage the service proxy, from which the
     *     proxied service itself can be obtained.
     */
    public IProxy createServiceProxy(IRemote impl, String facet);

    /** Tell the node how many concurrent threads it can start on the netlet's
     * behalf. For example, if we set the concurrency to 10 then the netlet
     * will only be able to accept 10 service requests, or 9 service requests
     * and 1 notification of termination. This does not affect how many
     * service requests the netlet can initiate. But if the netlet makes an
     * asynchronous call to a facet and the concurrency threshold is too low
     * then it may never receive the response. Since incoming calls are
     * otherwise out of the netlet's control, this method allows this resource
     * (threads) to have an upper limit.
     *
     * @param threads  Maximum number of threads the node can start for the
     *     netlet. Setting this to zero or less is silly, since it prevents
     *     the netlet from responding to any incoming request, including a
     *     notification of termination.
     *
     * @param thread_reuse  Can this netlet reuse threads? If a netlet invokes
     *     outwards synchronously then a causal incoming call was called
     *     synchronously then the original thread can be reused. Default is
     *     <tt>false</tt> (no thread reuse).
     */
    public void setConcurrency(int threads, boolean thread_reuse);
}
```

2.5 IRemote

The top level super-interface of anything that needs to passed through a mediator.

It is an empty marker interface with no methods. Among other things, all facets need to implement `IRemote`. When a netlet binds a facet it gets an `IRemote` object. It can then cast this into whatever type is actually needed.

2.6 IService

Represents a service connection between two netlets.

When a client netlet binds to a service each gets an `IService` interface in return. It also presents an `IService` interface to the service netlet.

Through this interface either one can terminate the relationship or bind to the other's facets.

```
public interface IService extends IFacetCollection
{
    /** Terminate this service connection. The node managing the connection
     * ensures that the connection is cut after the call is made and before
     * the implementing end receives the call.
     */
    public void terminate();
}
```