Technical Note:
JNBridgePro and Clustering

This technical note describes how JNBridgePro can be used as part of a clustered J2EE architecture, so that .NET code can participate in the failover, high-availability, and load-balancing capabilities offered by clustering. We first describe JNBridgePro's native failover capability, which does not depend on the clustering mechanism offered by any specific J2EE application server vendor, then we describe how to make JNBridgePro work with the proprietary clustering mechanisms offered by application server vendors.

JNBridgePro's failover capability

JNBridgePro offers a basic failover capability as part of its EE version that allows a .NET client to transparently switch its communications from one back-end Java server to another when the first one fails. Figure 1 shows the recommended architecture for using JNBridgePro failover to access an Enterprise Java Bean. Note that Java sides are deployed on each server in a WAR file that contains the JNBridgePro Java runtime component, an EJB client JAR file, and associated configuration information.

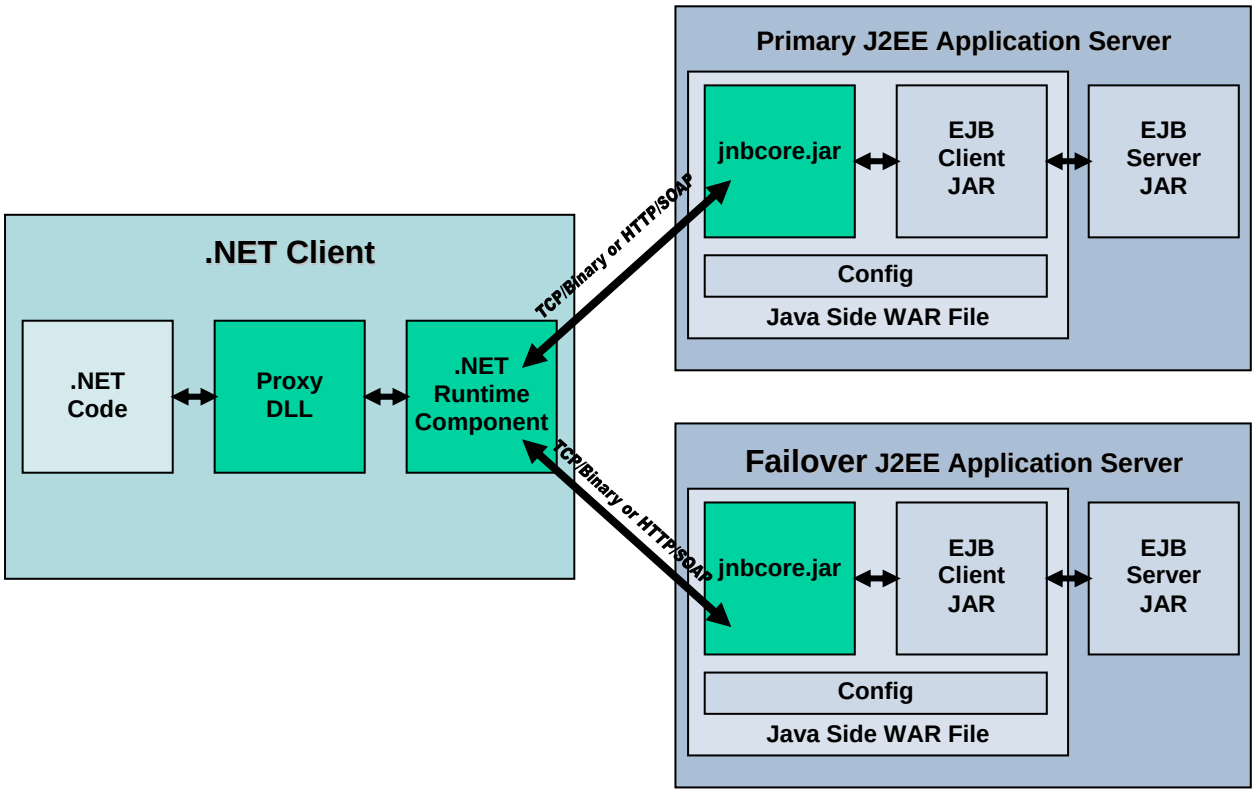

Figure 1: JNBridgePro Failover Architecture

When using failover, the JNBridgePro .NET-side configuration designates a *primary* Java side and a *failover* Java side. The primary Java side is the server that is used when the system starts up. When communication with the primary Java side fails, the JNBridgePro .NET side throws a FailoverException and switches the .NET-Java connection so that subsequent calls over JNBridgePro are sent to the failover Java side. Catching the exception offers the .NET-side client the opportunity to re-create session or state information on the failover server. As mentioned above, all subsequent calls over JNBridgePro are sent to the failover Java side until the communications mechanism is explicitly reset through the FailoverController.reset() API call.

Failover can be used in connection with JNBridgePro's multiple Java-side capability, in which case the .NET is configured so that each one of the Java sides being communicated with has a primary server and, optionally, a failover server.

JNBridgePro's failover capability has the advantage that it is not dependent on any on any clustering mechanism offered by any application server vendor. In fact, failover will work even if the Java side does not offer any clustering capability, and will even work with standalone JVMs in addition to J2EE application servers.

The failover capability does have some limitations. One cannot assume that the failover server will contain copies of the remote Java objects referenced by the .NET side that had resided on the failed primary server. (It is possible that the failover server was running in hot standby mode and was mirroring requests made to the primary server, but the failover mechanism does not assume this.) The .NET client may therefore need to re-create session and state information on the failover server, or even restart the session from scratch, when a FailoverException is received. In addition, if the primary server comes back up, communications will not be re-established with it until the connection is explicitly reset. Finally, the failover capability only supports failover functionality; it does not offer any other clustering-related capability, such as high-availability or load balancing. To get these additional capabilities, it is necessary to take advantage of vendor-specific clustering mechanisms as described in the next section.

For more information on failover, including configuration and the failover API, see the JNBridgePro *Users' Guide*.

JNBridgePro and vendor-specific clustering

If the user wishes to have .NET code participate in an application server's clustering mechanism, JNBridgePro can be used, but a different architecture must be used for the bridging. Figure 2 shows the architecture when accessing an Enterprise Java Bean in a clustered environment.

Note that the JNBridgePro Java sides are no longer deployed on the J2EE application servers, but rather are deployed on the client machines. Also note that the EJB client JAR files must be "cluster-aware": that is, the EJB client stubs must know how to participate in the J2EE app server vendor's clustering mechanism. It is generally the user's responsibility to assure that they are cluster-aware when the client stubs are generated.

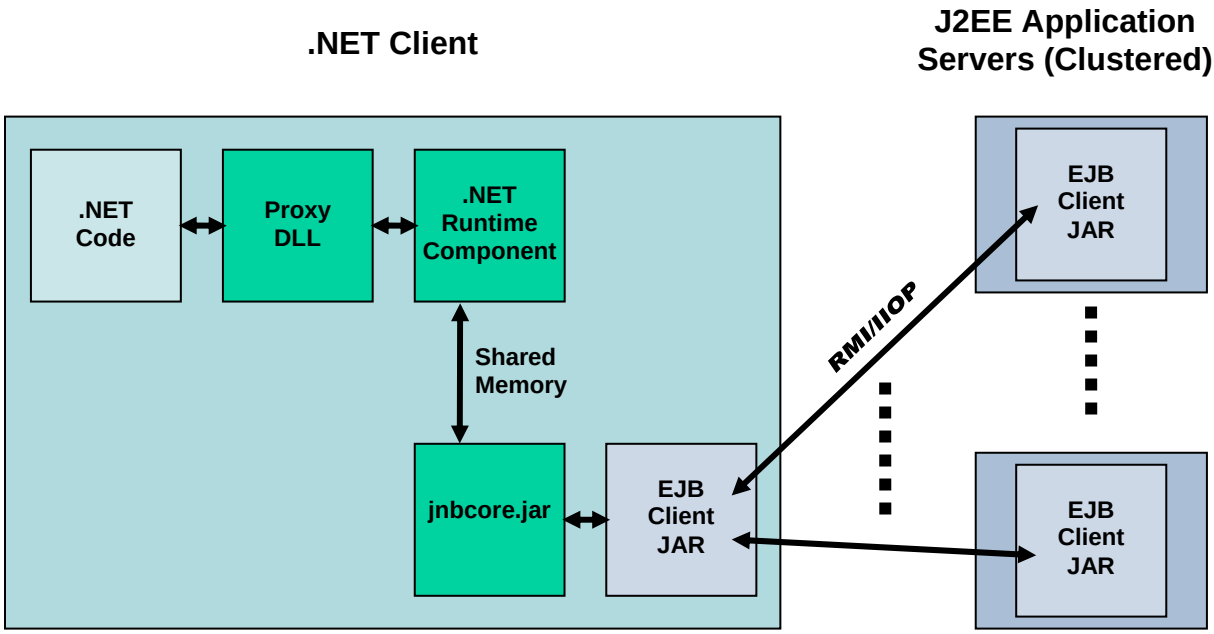

Figure 2: Accessing a EJB in a Clustered Environment

It should be noted that, in this architecture, the .NET/Java gap is bridged, and then communication between the EJB clients and server JARs occurs over the network. To minimize the effect of this extra hop, we recommend using the shared-memory communication channel between the Java and .NET sides as shown in the figure. Shared memory is supported starting with JNBridgePro v2.1.

Using this architecture, .NET client code can participate in all the clustering capabilities offered by the J2EE application server, including failover, high availability, and load balancing.

Migrating between JNBridgePro failover and vendor-specific clustering

Migrating from a JNBridgePro failover architecture to an architecture employing vendor-specific clustering is straightforward. Various components, including the JNBridgePro Java runtime component and the EJB client JAR file must be migrated from the server to the client machines, and configuration information must be changed to reflect the changes in the location of the various components. No changes need be made to the code. If the .NET client code was written to handle FailoverExceptions, this code may be left in since FailoverExceptions will no longer be thrown and the exception handler will no longer be executed. However, if the user wishes to remove this code as part of the migration process, it is safe to remove it.

Migrating from using vendor-specific clustering to JNBridgePro failover requires a bit more effort. In addition to migrating the various components from the client machine to the server, and modifying the configuration information to reflect these changes, the user must write code to catch the FailoverExceptions, which likely will include code to re-create session and state information on the failover server. The user may also need to write code to explicitly reset the failover mechanism to communicate with the primary server when it is brought back up.

JNBridge, LLC
www.jnbridge.com