www.psinaptic.com

PsiNaptic

PsiNaptic Inc.
216, 200 Rivercrest Drive SE
Calgary, Alberta
Canada T2C 2X5
T: 403 720-2531
F: 403 720-2537

The question is "If there is no RMI how can I interact with your JMatos service?"

The response below is intended to be as simple as possible, but also accurate.

JMatos is a Technology Compatibility Kit (TCK) compliant implementation of the Jini Lookup Service (LUS). We have run and passed the latest TCK from Sun. What this means is that if you are a Service Provider or a Service Consumer, you should not detect any difference between a JMatos implementation of the Jini LUS and the Sun Microsystems's reference implementation of the Jini LUS (commonly referred to as "Reggie", after the jar file name).

There are 2 main actions associated with a Jini LUS. These are;
1. Discover LUSs on the network
2. Joining an LUS (registering or obtaining a service with/from the discovered LUS).
These protocols are defined by the Jini specification [1].

The Discovery protocols use a combination of multicast and unicast TCP/IP datagrams to discover the existence of and the address of Jini lookup services. For the most part these protocols do not use RMI functionality to fulfill their function, and are therefore not relevant to this discussion. However, the final result of the discovery protocol is an **`java.rmi.MarshalledObject`** that is returned to the requestor (either a Service Provider or Consumer).

The Join protocol can be described simply as obtaining an object from the discovered LUS that is of class type ServiceRegistrar. This class is contained in the above mentioned **`MarshalledObject`** Using this object, the Service Provider or Service Consumer can then use the standard calls provided by the ServiceRegistrar class to register with the LUS or query for services already registered with the LUS (represented by the ServiceRegistrar).

The Reggie implementation of the Jini LUS uses standard RMI calls to return the ServiceRegistrar from the Jini LUS to the requestor. Further, Reggie does not return to the requestor a ServiceRegistrar object, rather it returns an RMI proxy of the ServiceRegistrar. Reggie then uses RMI calls back to the LUS through the ServiceRegistrar object to fulfill the functionality defined in the Jini specification.

JMatos differs here from the Reggie reference implementation of the Jini Specification in a couple of ways, while maintaining compliance with the specification. Firstly, JMatos returns a byte compatible ServiceRegistrar MarshalledObject to the requestor, without using RMI classes. From the point of view of the requestor, the ServiceRegistrar can be unmarshalled and instantiated exactly the same way from JMatos as from any other implementation. To the requestor, it makes no difference if the LUS discovered is JMatos or Reggie. Secondly, the ServiceRegistrar returned from JMatos is **not** an RMI proxy object. The ServiceRegistrar returned from JMatos is a fully functioned, self-contained object that is capable of fulfilling all ServiceRegistrar (LUS) functions as defined in the Jini specification [1] locally. In both cases (JMatos or Reggie), the implementation of **how** the ServiceRegistrar implements registration, service matches, etc is hidden from the client, therefore it makes no difference if that functionality uses RMI or not.

The consequences of these differences are significant. Where a Reggie implementation requires the RMI infrastructure (RMI activation, RMI registry, etc) and significant resources to support this infrastructure, JMatos does not.

A related question than is often asked is "If JMatos does not use RMI can I still register RMI proxies as services?" The answer is YES. The Jini LUS (Reggie or JMatos) don't ever instantiate or unmarshall a service. The byte code representing the service is simply held in the LUS and provided to any Jini client that requests that service. In both cases the LUS never knows or cares if the service is an RMI proxy or whether it is a self-contained (non-RMI) class.

It should be noted that the Service Provider and Service Consumer must know what kind of service they are providing and getting from the LUS. If a Service Provider registers an RMI proxy as a service and a Service Consumer obtains that service, the Service Consumer must have RMI client capabilities while the Service Provider must have RMI server capabilities for that service to function correctly. The determination of whether a service is RMI based is easily ascertained since they must extend **`java.rmi. Remote`**. The determination of whether a Jini application (set of services) should be RMI based or not is a separate and distinct application architecture question, and is not related to the LUS itself.

In conclusion, JMatos was designed and coded to meet the Jini specification. The requirement to get a Jini LUS as small as possible outweighed the convenience of using RMI to hide the TCP/IP network.
Sun's Reggie implementation of a Jini LUS is simply that – an implementation of a published specification and like all implementations must pass the TCK tests. Both JMatos and Reggie have passed the TCK and can coexist on the same network with Jini clients not realizing or caring. Probably the best example of how this works is to get JMatos and Reggie running and simply use the Jini Browser (provided in the Jini SDK) to query the separate lookup services. There will be no difference except in complexity and size.

[1] Arnold, K., Waldo, J., "The Jini Specifications", 2nd Ed, Upper Saddle River, NJ, 2001.