

Project Shoal – A Generic Clustering Framework

Shreedhar Ganapathy(GlassFish)

shreed@sun.com

Mohamed Abdelaziz(JXTA)

hamada@jxta.org



What is Project Shoal?

- A Java.net project aimed at building a Clustering Framework
 - > for Java EE/J2EE Application Servers and any other product requiring clustering features
 - > At <https://shoal.dev.java.net>
- Shoal provides a Group Management Service (GMS) that provides
 - > group membership management through discovery of events
 - join, shutdown and failure notifications, delegated recovery initiation, and
 - > state caching facilities
- Applications interact with Shoal's GMS API using their logical identity semantics to communicate with other group members

Shoal GMS Feature Themes

Three broad feature themes:

- Features providing
 - > a group sensory-action theme.
 - > a group communications theme.
 - > Shared or Distributed Storage theme.

Shoal GMS Group Sensory-Action features

- Provides a set of Client APIs for signalling cluster events. Such Signals include
 - > Lifecycle Signals
 - > Cluster Member(s)
 - joining the cluster at runtime
 - leaving the cluster at runtime
 - going into in-doubt(suspected) state.
 - being confirmed failed
 - > Recovery oriented Signals and Support
 - > Automatic Recovery Member Selection Signal
 - > Protective failure fencing operations

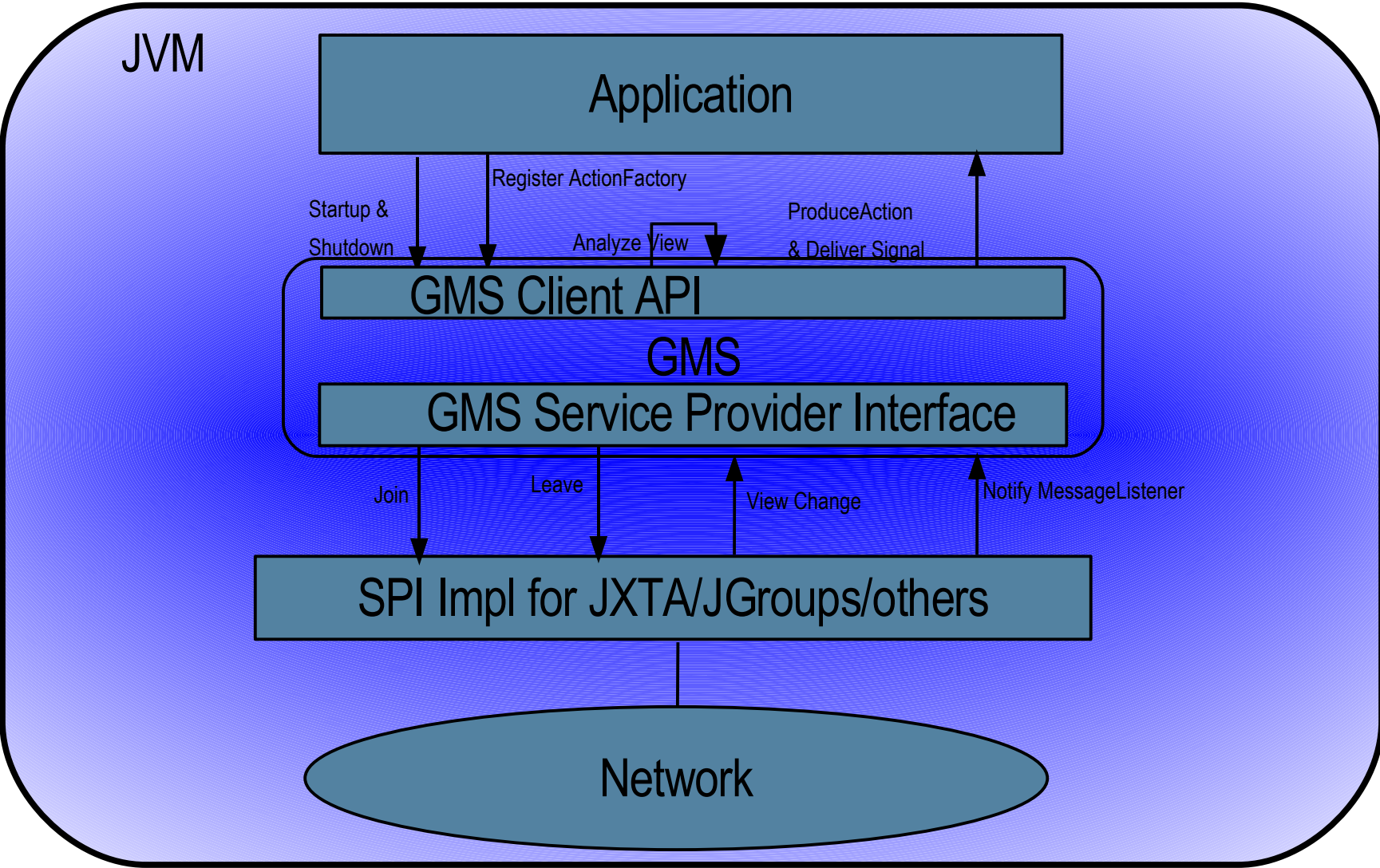
Shoal GMS Group Communication Theme

- GMS provides Group Communication Provider SPI
 - > Group communication technologies such as JXTA, JGroups, etc. integrate through SPI
- GMS provides a group messaging handle
 - > to clients to send messages to group or particular member(s),
 - > client components can address messages to specific components in destination
- GMS hands Message Signals in recipient clients
 - > GMS delivers the Signal to the target component

Shared or Distributed Storage Theme

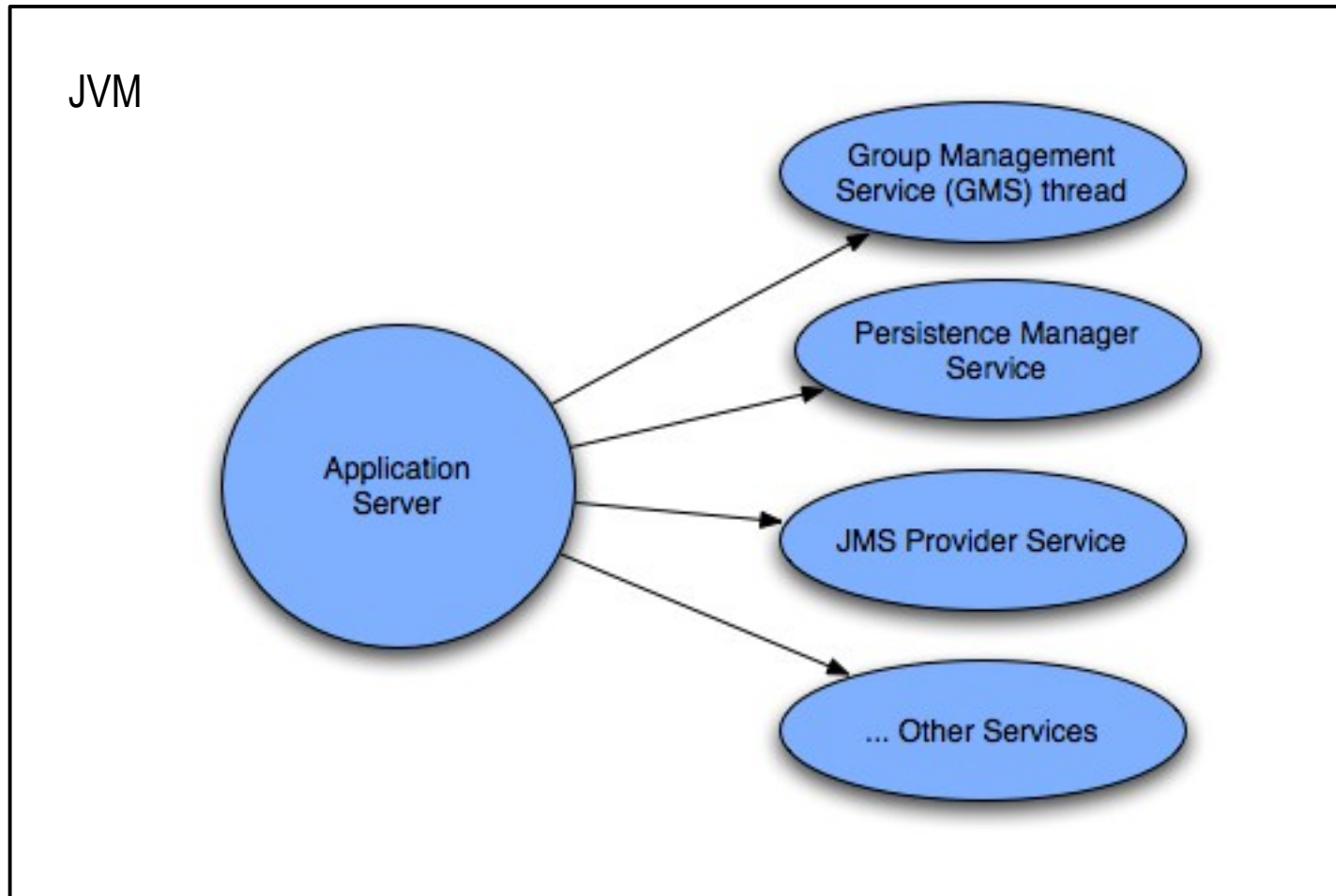
- GMS provides a Distributed State Cache (DSC) interface
 - > Can be implemented to suit custom requirements
 - > Default Implementation is a shared concurrent hashmap
- DSC can be implemented for in-memory shared/distributed cache for application state
- Group communication providers provide tunable performance properties for better throughput

Application, Shoal GMS, Group Communication Provider Relationships



Shoal GMS in Application Server Instance

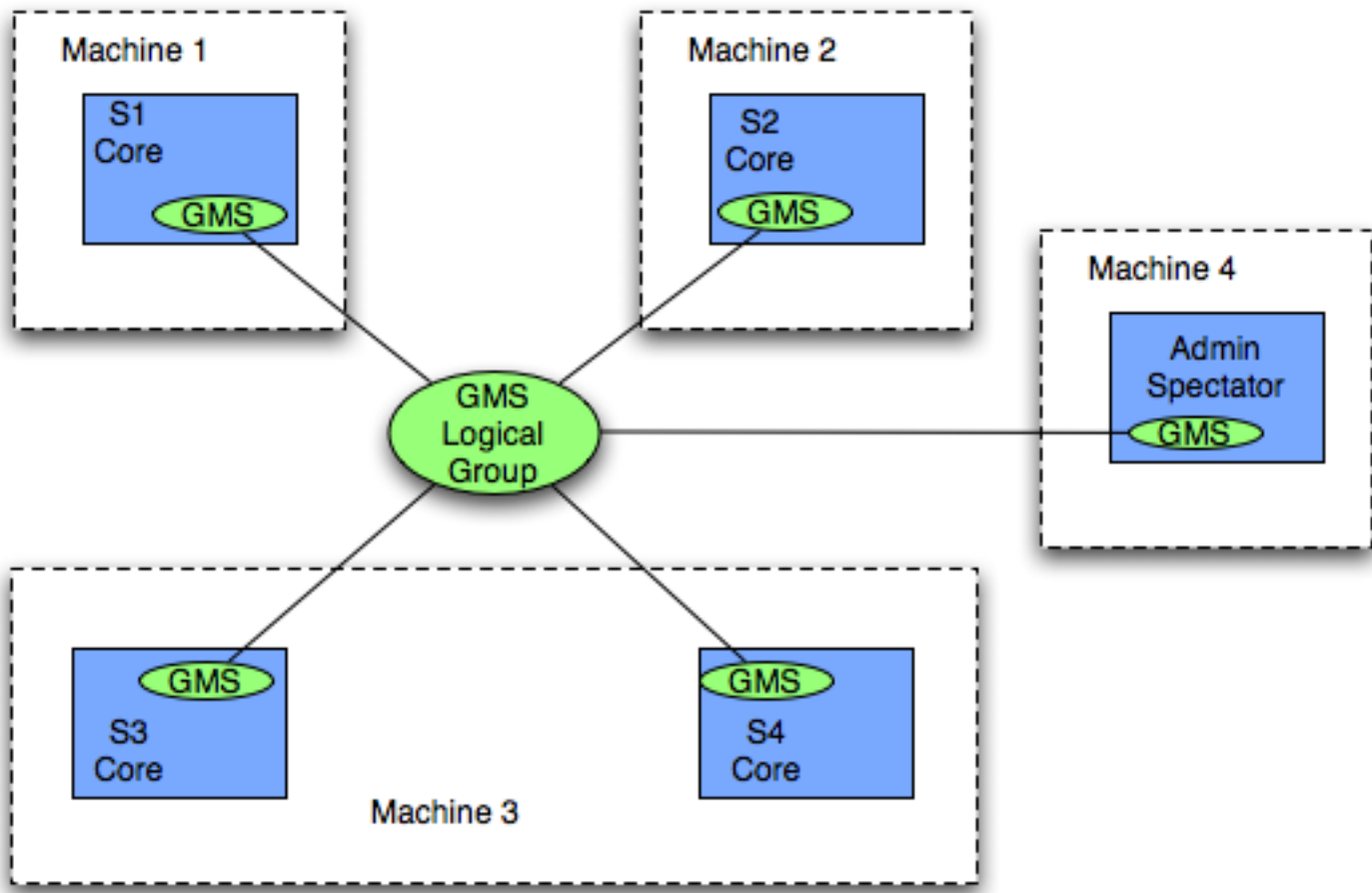
Application Server starts various in-process Services, one of which is the GMS



Shoal Group Management Service

- At startup, GMS in each process joins predefined group (and at shutdown leaves that group).
- Pluggable GroupCommunicationProvider Impl provide communication channels, and protocols for group composition and failure monitoring
- When member joins, leaves or fails, GMS calls client components informing them
- On failure confirmation, if enabled, Recovery Oriented Computing Support kicks in –
 - > GMS selects a recovery candidate member
 - > Notifies registered client components in selected member process
 - > Shares this selection information through DSC.
 - > Protects recovery operations through failure fencing
 - > Ensures recovery-in-progress ops are tracked for multiple failures
- Provides a default Distributed State Cache implementation for caching recovery states and application lightweight data

Shoal GMS in an Application Server Cluster



What do Shoal GMS clients get?

- Peace of mind :)
- Saves many person years of work in writing complex code to emulate its functionality in common enterprise applications
- GMS takes on the complexity of group formation, discovery of members, preconfigured endpoint locational details, networking semantics
- Clients simply use the group's logical membership identities to communicate and be notified of events

What do Shoal GMS Clients get ?

- Allows client components in a process to :
 - > Send and Receive Messages using app level addressing semantics ex. Using instance Id or name for addressing the destination.
 - > Use GMS Event Model for receiving Group Event Notifications & Message Delivery
 - > Use GMS APIs for getting member states, current group composition, caching app level information, and for messaging one-to-one, one-to-many, and one-to-all.
- Each system installation uses a particular Group Communication Provider, plugs in the same with SPI implementation. Clients don't change any code.
- Useful features yet a lightweight component providing an engine for building enterprise distributed systems functionality
- Recovery oriented computing semantics without application specific artifacts, a basis for building fault tolerance solutions.
- Several current use cases within Sun's Appserver, more to come...

Shoal GMS Startup code sample

```
public class GMSLifecycleManager {
    Runnable gms;
    public void startGMS(){
        try {
            //creates a Runnable and inits with serverId, groupId, membertype and lifetime
            //config properties.
            gms = GMSFactory.startGMSModule(serverId, groupName,
                GroupManagementService.MemberType.CORE, properties);
            Thread t = new Thread(gms, "GMSThread");
            t.start();
        }
        catch (GMSException e){
            //deal with it :)
        }
    }
    public void shutdownGMS(){
        gms.shutdown(GMSConstants.ShutdownType.INSTANCE_SHUTDOWN)
    }
}
```

Shoal GMS Client CodeSample

```
public class GMSClient implements CallBack {  
    ....  
    registerWithGMS(){  
        GroupManagementService gms = GMSFactory.getGMSModule(clusterName);  
        //register interest in events  
        gms.addActionFactory(new JoinNotificationActionFactoryImpl(this));  
        gms.addActionFactory(new FailureSuspectedActionFactoryImpl(this));  
        gms.addActionFactory(new FailureNotificationActionFactoryImpl(this));  
        gms.addActionFactory(new FailureRecoveryActionFactoryImpl(this));  
        gms.addActionFactory(new PlannedShutdownActionFactoryImpl(this));  
    }  
  
    processNotification(Signal signal){  
        //process the appropriate Signal type, say FailureNotificationSignal according to client logic  
    }  
}
```

As seen above, for GMS clients, this is a Breeze to do and very simple. GMS takes on complexities of Group and Endpoint discovery, failures, etc.

Shoal GMS in GlassFish V2

- In GlassFish v2 cluster mode, Shoal GMS is used for
 - > Automated delegated transaction recovery
 - > Timer migrations
 - > IIOP Failover Loadbalancer
 - > Self Management
 - > Read-only Bean's cache change notifications
 - > Domain Admin Server for cluster health
 - > In-memory replication component's discovery and failure detection needs.

Shoal GMS in the enterprise world

- Shoal can be used for common enterprise clustering requirements
- Some products that can benefit
 - > MQ Broker Clusters
 - > Directory Server Clusters
 - > Sun Grid
 - > App level clustering in small scale deployments (plug in Shoal into a PE instance and apps directly use it for their cluster needs)
 - > Several others limited only by imagination and some contrarian thinking :)

GMS SPI Highlights

- Goaled to work with both JGroups and JXTA
- Extracted out of common functionalities from both the group communication technologies and GMS client requirements
- Open to other GCP implementations as suitable for a specific application
- SPI rev in progress

GMS's Use of JXTA

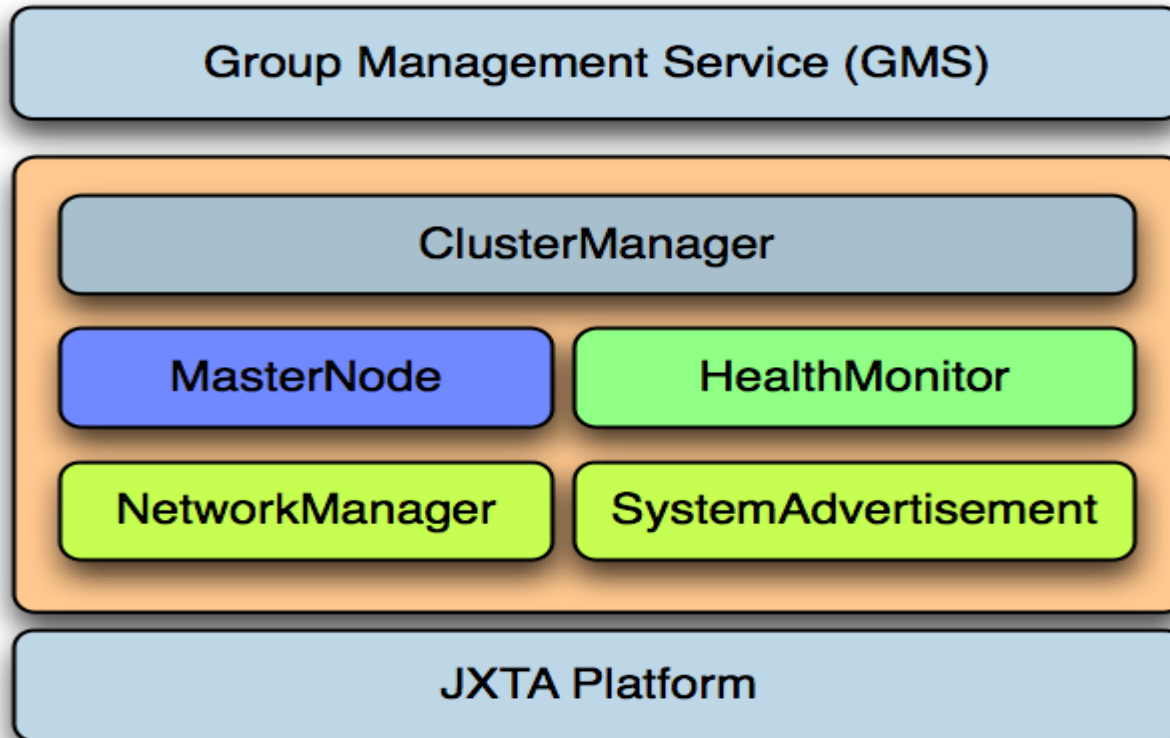
- GMS requirements
- Jxta Management – a collaborative effort between Appserver Group and JXTA (Advanced Development) Group
- Critical JXTA Platform Functionality

GMS Requirements

- At the minimum
 - > Group and Membership detection
 - > Failure Detection
 - > Guaranteed Message delivery
 - > Ordered Messaging (particularly for group membership messages)
- Of Added Use
 - > Flow Control (Dynamic Sliding Window management)
 - > Merging of split groups
 - > Fragmentation of packets over 64K

JxtaManagement Architecture

- Shoal GMS utilizes JxtaManagement component (a JXTA based group service provider) for dynamic cluster configuration, formation, and monitoring.



JxtaManagement Architecture

- **NetworkManager**
 - > Given instance and group name, uses a SHA-1 hash to encode the cluster GroupID, and NodeID
 - > defines a set of predefined communication identifiers used for formation, monitoring and messaging.
 - > Application can pass additional config parameters, such as bootstrapping addresses to facilitate cross sub-net and firewall communication.
- **SystemAdvertisement**
 - > An extensible XML document describing system characteristics (HW/SW configuration. CPU load would be a nice extension).
 - > Envisioned that this would serve at the foundation of a Grid framework.
- **MasterNode**
 - > Lightweight protocol allowing a set of nodes to discover one another, and autonomously elect a master for the cluster.
 - > Resilient to multi node collisions and employs an autonomous mechanism to avoid network chatter to resolve collisions.

Jxta Management Architecture

- ClusterView
 - > Maintains an ordered view of the Cluster
- HealthMonitor
 - > A lightweight protocol allowing a set of nodes to monitor the health of a cluster.
 - > Relies on a tunable heart beat,
 - > acted upon by MasterNode to notify the group of failures,
 - > and by other members to elect a new master if the master node fails.
- ClusterManager
 - > Manages lifecycle of this SPI

Critical Jxta Platform Functionality

- Membership scoping - Infrastructure NetPeerGroup provides group isolation from the world
- Rendezvous Protocol - PeerGroup and Peer locational and route tracking, and provides end point routing abstraction
- Platform provides virtualizing of PeerID to network addresses
- Platform's messaging envelope - the Message object encapsulates MessageElements allowing for separation of payload from metadata
- Secure communication channels – PKI-based public key for Unicast, shared keystore based for multicast
- NetworkConfigurator – API for programmatic configuration, configuration stays in-memory during lifetime of peer.

Current Status, Tests Run

Current Status

- Source code has been made available at Project Shoal. Download it and have fun with it :)
- GMS SPI implementation uses Jxta layer implementing Group Communication Provider SPI
- Weekly review meetings with JXTA team for continuous improvement

Tests Run

- Tests covered: Various time Startup scenarios, Join tests, Shutdown tests, Failure tests, and Recovery behaviors tests
- QE ran 15 most important test cases from GMS suite of 40 with several iterations.
- Fixing P3s as they are identified.

Plans

- Stabilize current implementation for release with GlassFish Application Server 9.1
- Involve user community to test and deploy Shoal and contribute bugs and RFEs.
- Possible Shoal Cache implementation being looked at.
- Engage within and outside Sun for adoption.
- **Use Shoal as a driver for GlassFish downloads.**