Dynamic Runtime Clustering in Glassfish v3.1 using Shoal Group Management Service

One pager template version: 1.9

1. Introduction
    1.1. Project/Component Working Name:
           Dynamic Runtime Clustering Support via Shoal Group Management Service

    1.2. Name(s) and e-mail address of Document Author(s)/Supplier:
           Name: joe.fialli@oracle.com
           Name: bobby.bissett@oracle.com

    1.3. Date of This Document:
           07/13/2010 incorporate changes to 4.5 interfaces for AliveAndReadyCoreView management.
           06/30/2010 incorporate changes 4.5 Interfaces changes to complete implementing split of shoal-gms into shoal-gms-api and shoal-gms-impl jars.
                       Updated GroupManagementServer.getPreviousAliveAndReady() based on design discussion with Mahesh.
           06/11/2010 updated access to GMS within GF v3.1 after design session with Jerome
           06/02/2010 updated based on Mahesh's feedback
           05/24/2010

2. Project Summary
    2.1. Project Description:
         Integrating Shoal Group Management Service into a Glassfish cluster allows

         * other Glassfish application services to register for notification of other clustered instances joining and leaving
           (normal and abnormal termination) the glassfish cluster
         * get the current member status of any member in the cluster.
         * sending a message to one, some or all clustered instances in the cluster
         * get a list of Core or All clustered instances in the glassfish cluster.

    2.2. Risks and Assumptions:
         Miss reporting GMS FAILURE notification on fast restart(instance restarted faster than GMS default heartbeat failure detection configuration)
         due to no NodeAgent in GF v3.1. See details in section 4.12

discussion on how application is notified of missed GMS
notification of FAILURE due to fast restart. If GF v3.1, a clustered instance could be restarted faster than
GMS heartbeat failure detection time due to clustered instance being a registered native OS service that is automatically
restarted when the native OS detects that the service has failed.

Metro Reliable Messaging(RM) HA requires larger payloads than GMS messaging supported in past releases.
For session data, payloads were only in the range of 2k to 10k. For Metro Reliable Messaging RM,
the message payload size is application dependent and can range significantly in value.
We are evaluating larger message payloads for GMS sendMessage in Shoal GMS distributed unit level
testing now to limit future risk. We are working with Metro RM team to identify a ballpark range of size and msg
throughput.

3. Problem Summary

3.1. Problem Area:

GMS provides infrastructure to build fault tolerance, reliability and availability within a glassfish cluster.

3.2. Justification:

High availability of session data in a glassfish cluster is a release driver for v3.1 and the
High availability of session data(i.e., http sessions, stateful ejb) is built on top of GMS subsystem.
The GMS subsystem provides high availability module with the ability to send messages within the cluster and
the ability to compute a consistent hash based on the clustered CORE instances that are currently running.

Other cluster services using GMS include IIOP, IIOP load balancer, EJB timer migration, delegated transaction recovery, Metro RM.

4. Technical Description:

4.1. Details:

Shoal GMS over JXTA transport was used by high availability in Glassfish v2.x.
For Glassfish v3.1, Shoal GMS over Grizzly transport is going to be used.
The motivation for changing from jxta transport to grizzly transport were numerous.
Grizzly is already part of the Glassfish v3. Grizzly is actively being optimized and
further developed. The Grizzly developers are part of Glassfish

development and
are able to provide support for the transport.

We have done Shoal GMS unit testing to verify that Shoal GMS over Grizzly is as predictable and
performant or higher performing than Shoal GMS over JXTA. These Shoal GMS unit test include
message sending throughput and validating GMS notifications are working as well with
Shoal over Grizzly as it does with Shoal over Jxta in past.

We have written distributed Shoal GMS unit test to simulate HA messaging patterns and throughput.
This allowed us to validate that Shoal GMS messaging is sufficient for HA needs.

When does an application server instance joins a GMS group.
1. When a clustered instance belongs to a cluster with gms-enabled with a value of true, the clustered instance
joins the cluster when cluster instance is started.
(see http://wiki.glassfish.java.net/attach/V3FunctionalSpecs/gmsconfig_gfv3_1.rtf for more on
specification of domain.xml element cluster attribute gms-enabled).

2. When a DAS is started, it joins each of its cluster that has gms-enabled with a value of true.

4.2. Bug/RFE Number(s):
// List any Bug(s)/RFE(s) which will be addressed by this proposed change.
// Provide links to the Bug(s)/RFE(s)where possible.
// RFE's must be trackable via an issue in Issue Tracker for
// features in the open source distro and in Bugster for value-add
// features to be released in the commercial distro.

TBD: Will enter Milestone Features listed in http://wiki.glassfish.java.net/Wiki.jsp?page=GlassFishv3GMS into issue tracker and reference them.

4.3. In Scope:
// Aspects that are in scope of this proposal if not obvious from above.
Support for clustered instances of a glassfish cluster are required to be on same subnet and multicast needs
to be enabled for the subnet on each machine and router connecting the machines. This requirement was required
in glassfish v2.1.1.

4.4. Out of Scope:

// Aspects that are out of scope if not obvious from above.

Virtual Multicast Support (substitute UDP broadcast with a static list of IP Addresses and ports).

Missed FAILURE notifications due to quick restart of a failed clustered instance.

In gf v2.1.1 the Nodeagent notified GMS (via GMS watchdog failure notification) when it was going to restart an instance. Since there is no NodeAgent in GF v3.1, this is no longer possible.

See details of how this worked in GF v2.1 in following document: http://wiki.glassfish.java.net/attach/SFv2FunctionalSpecs/gmsWatchdog.doc

The REJOIN subevent described in section 4.5.1 addresses how a GMS client will be notified of missed GMS notifications of FAILURE when a clustered instance is restarted very quickly.

GMS client is responsible for checking for REJOIN subevent on JOIN and JOINED_AND_READY events as a means to detect fast restart of clustered instance.

4.5. Interfaces:

// Interfaces may be commands, files, directory structure, ports,
// DTD/Schema, tools, APIs, CLIs, etc.
// Note: In lieu of listing the interfaces in the one pager, providing
// a link to another specification which defines the interfaces
// is acceptable.

4.5.1. Public Interfaces

An EJB or web tier application attains access to GMS via GMSFactory.getModule().

****

GF v3.1 modules access GMS via following api,

// List new, public interfaces this project exports.

Module glassfish/v3/cluster/gms-bootstrap depends on shoal-gms-api.jar only.

@Service
class GMSAdapterService // loads gms-adapter module only if gms-enabled.

```
                /**
                 * Return the GMSAdapter for a clustered instance in a
cluster with gms-enabled is true.
                 * Works in DAS with one cluster.
                 * Throws IllegalStateException if called when DAS has
multiple clusters.  Use getGMSAdapterByName(String) when in DAS specific
code.
                 */
                GMSAdapter getGMSAdapter();

                /**
                 * Return the GMSAdapter for a cluster named "clusterName"
with gms-enabled is true.
                 * Strongly recommend using this method in DAS specific
code since DAS can have more than one cluster.
                 * A clustered instance can belong to one and only one
glassfish cluster.
                 */
                GMSAdapter getGMSAdapterByName(String clusterName);   //
must be used in DAS when it has multiple clusters.


                boolean isGmsEnabled();


            Module glassfish/v3/gms-adapter

            import com.sun.enterprise.ee.cms.core.CallBack;

            @Contract
            interface GMSAdapter {
                   /** Get implementation of GroupManagementServicefrom
Shoal GMS API  module needs to add dependency on shoal-gms-api-1.5.3(or
higher).jar
                    * @returns GroupManagementService.
                    */
                   GroupManagementService getModule();
                   String getClusterName();

                   // use the following methods in GF v3.1 rather than
GroupManagementService.addActionFactory(...)
                   // The default client implementations of
ActionFactoryImpl are not in shoal-gms-api*.jar.
                   void registerJoinNotificationListener(CallBack
callback);
                   void registerJoinedAndReadyNotificationListener(CallBack
callback);
                   void registerMemberLeavingListener(CallBack callback);
                   void registerPlannedShutdownListener(CallBack callback);
```

```
                    void registerFailureSuspectedListener(CallBack
callback);
                    void registerFailureNotificationListener(CallBack
callback);
                    void registerFailureRecoveryListener(String
componentName, CallBack callback);
                    void registerMessageListener(String componentName,
CallBack messageListener);
                    void
registerGroupLeadershipNotificationListener(CallBack callback);

                    // use the following methods rather than
GroupManagmentServer.removeActionFactory(...)
                    void removeFailureRecoveryListener(String
componentName);
                    void removeMessageListener(String componentName);
                    void removeFailureNotificationListener(CallBack
callback);
                    void removeFailureSuspectedListener(CallBack callback);
                    void removeJoinNotificationListener(CallBack callback);
                    void removeJoinedAndReadyNotificationListener(CallBack
callback);
                    void removePlannedShutdownListener(CallBack callback);
                    void removeGroupLeadershipLNotificationistener(CallBack
callback);
                    void removeMemberLeavingListener(CallBack callback);
            }
```

Other modules in Glassfish V3.1 that require access to GMS can access it in following way:

GMS Client access to GMSService in clustered instance and in DAS with only one cluster.

```
                import org.glassfish.gms.GMSAdapter;
                import org.glassfish.gms.bootstrap.GMSAdapterService;
                @Inject
                GMSAdapterService gmsAdapterService;
                GroupMangementService gms;

                postConstruct() {
                     if (gmsAdapterService.isGmsEnabled()) {
                         gms =
gmsAdapterService.getGmsAdapter().getModule();
                     }
                }
```

Add methods to existing Shoal GMS class com.sun.enterprise.ee.cms.core.GroupHandle

```
    /**
     * This snapshot was terminated by AliveAndReadyView.getSignal(), a GMS
notification of
     * either JoinedAndReadyNotificationSignal, FailureNotificationSignal
or PlannedShutdownSignal.
     *
     * <p>
     * Behavior is not well defined at during GROUP_STARTUP or
GROUP_SHUTDOWN.
     *
     * <p>
     * If the last GMS notification was a JOIN with a REJOIN subevent, the
list previous CORE members will be same as list of current CORE members.
     * (this scenario reflects a fast restart of an instance in less than
GMS heartbeat failure detection can detect failure, this is called a REJOIN
     * when an instance fails and restarts so quickly that
FAILURE_NOTIFICATION is never sent. THe REJOIN subevent represents the
unreported FAILURE
     * detected  at the time that the instance is restarting.)
     *
     *
     * @return the previous AliveAndReady Core member snapshot
     */
    AliveAndReadyView getPreviousAliveAndReadyCoreView();   // to be used
by HA module

    /**
     * Get a current snapshot of the AliveAndReady Core members.
     * AliveAndReadyView.getSignal() returns null to signify that
     * the view is still the current view and no GMS notification signal
     * has terminated this current view yet.
     *
     * @return current view of AliveAndReady Core members
     */
    AliveAndReadyView getCurrentAliveAndReadyCoreView();


********

    new Interface AliveAndReadyView for package
com.sun.enterprise.ee.cms.core;

   /**
    * A read-only view consisting of all the AliveAndReady CORE members of
a GMS group.
    *
    * The GMS notification signals of JoinedAndReadyNotificationSignal,
FailureNotificationSignal and PlannedShutdownSignal
    * transition from one of these views to the next view. When one of
```

```
these signal occurs, the current view is terminated
    * by setting its signal.  While the view's signal is null, it is
considered the current view. Once a terminating signal
    * occurs, than this view is considered the previous view and
getSignal() returns the GMS notification that caused this
    * view to conclude.
    */
   public interface AliveAndReadyView {

      /**
       * These are members of this view BEFORE the GMS notification signal
that terminated
       * this view as being the current view.
       *
       * @return an unmodifiable list of sorted CORE members who are alive
and ready.
       *
       */
      SortedSet<String> getMembers();

      /**
       *
       * @return signal that caused transition from this view. returns null
when this is the current view
       *         and no signal has occurred to cause a transition to the
next view.
       */
       Signal getSignal();


      /**
       *
       * @return time this view ceased being the current view when its
signal was set.
       */
      long getSignalTime();

      /**
       * Monotonically increasing id.  Each GMS notification signal for a
core member that causes a new view to be created
       * results in this value being increased.
       * @return a generated id
       */
      long getViewId();

      /**
       * @return duration in milliseconds that this view is/was the current
view.
       *          If <code>getSignal</code> is null, this value is still
growing each time this method is called.
```

```
       */
      long getViewDuration();


      /*
       * @return time that this view got created.
       */
     long getViewCreationTime();
  }


            *********
            The following functionality is only necessary when gms group
members are restarted immediately after a software failure.
            This functionality is needed in Glassfish v3.1 since there will
be a local machine mechanism where a failed
            clustered instance will be automatically restarted (registered
as a native OS service).


            Add interface for Rejoin subevent to GMS Join and
JoinedAndReady Notificiation signal.

            Add interface com.sun.enterprise.ee.cms.core.RejoinSubevent.

            /**
             * Representation of a missed FAILURE notification when
restarting an instance.
             */
            interface RejoinSubevent {
                // time that the failed instance instantiation joined the
group.
                long getGroupJoinTime();
           }

            interface RejoinableEvent {
                // Returns RejoinSubevent if this Join or JoinedAndReady
notification is for an instance that restarted quicker
                // than GMS heartbeat failure detection was able to report
the GMS notification FAILURE.
                // Returns NULL if this instance is not restarting without
gms group being notified that it had left group in past.
                RejoinSubevent getRejoinSubevent();
            }

            Existing com.sun.enterprise.ee.cms.core JoinNotificationSignal
and JoinedAndReadyNotificationSignal will implement RejoinableEvent.

            We were able to avoid adding this functionality in Glassfish
v2.1.1 due to technique described in this
```

document: http://wiki.glassfish.java.net/attach/SFv2FunctionalSpecs/gmsWatchdog.doc.

Rationale on why GMS did not just issue a FAILURE notification followed by a Join notification when an instance restarted faster than GMS heartbeat failure detection could detect the instance failed. This rationale justifies why REJOIN is being introduced.

- When a fast restart occurs, GMS notices that a new instantiation of a member has started and GMS still had a reference to the previously failed instance. At this point in time, if one calls GroupHandle.getMemberStatus() on the restarted instance, one would get the response that it was alive. So it was considered too late to send a FAILURE notification for a member when it already had restarted. The REJOIN subevent on JOIN and JOINED_AND_READY GMS notifications was the chosen way to indicate to the application that a previous instantiation of member had FAILED while a new instantiation of the member is JOIN'ing the cluster.

Detail description of the fast restart is described in Section 3.1 of http://wiki.glassfish.java.net/attach/SFv2FunctionalSpecs/gmsWatchdog.pdf.
The REJOIN subevent indicates that an instantiation of the instance that joined the cluster at a certain time in the past has failed.
The REJOIN subevent has the time the previous instantiation of member had joined the cluster. The Shoal GMS log events also report that no FAILURE event was sent for the FAST RESTART of the member. The actual log events are in gmsWatchdog.pdf referenced above.
|

****

Mahesh's review requested this functionality. Below are my initial thoughts. Need assistance on next steps.

TBD: Add a capability for GMS notification callback handlers to be found by hk2 and automatically registered.
Require assistance of someone who understands hk2 on what impact this has on gms-adapter implementation

We will either need to add an annotation to the callbacks or potentially introduce marker Callback interfaces for each GMS notification in GMS. (i.e. a JoinCallback that extends com.sun.enterprise.ee.cms.core.Callback, one for JoinedAndReady, and so on.)
Currently, there is only one generic Callback for all GMS notifications, com.sun.enterprise.ee.cms.core.Callback.
The developer implements an implementation of this callback which contains a processNotification(Signal).
The current API allows a Callback to be registered for one

or more GMS notifications.

The following lines are used to specify which GMS notifications callbacks should be called for.
Notice that there are two distinctly different coding patterns that a Shoal GMS developer may use
when writing Shoal GMS callbacks.

In Shoal GMS test HAMessageReplicationSimulator.java, there is one callback per notification type.

```
gms.addActionFactory(new JoinNotificationActionFactoryImpl(new JoinNotificationCallBack()));
gms.addActionFactory(new JoinedAndReadyNotificationActionFactoryImpl(new JoinAndReadyNotificationCallBack(memberID)));
gms.addActionFactory(new MessageActionFactoryImpl(new MessageCallBack(memberID)), "TestComponent");
```

While in Shoal GMS test ApplicationServer and GMSClientService "simulation", there is only one callback implementation registered
with many different GMS notification callbacks.

```
class GMSClientService implements com.sun.enterprise.ee.cms.core.Callback {
 public GMSClientService(final String serviceName, final String memberToken){
    gms = GMSFactory.getGMSModule();
    gms.addActionFactory(new PlannedShutdownActionFactoryImpl(this));
    gms.addActionFactory(new JoinNotificationActionFactoryImpl(this));
    gms.addActionFactory(new FailureNotificationActionFactoryImpl(this));
    if (memberToken != null && memberToken.compareTo("server") != 0) {
      gms.addActionFactory(serviceName, new FailureRecoveryActionFactoryImpl(this));
     gms.addActionFactory(new MessageActionFactoryImpl(this), serviceName);
    }
    gms.addActionFactory(new JoinedAndReadyNotificationActionFactoryImpl(this));
  }

  public void processNotification(Signal signal) { .... /* process GMS signal notifications */ }
```

Do we select one style or do we have to accomodate both

styles in Glassfish v3.1 hk2?

When a callback is registered, application dependent data is sometimes associated with the callback.
Would an GMS spefic annotation be needed to accomplish this in hk2, or is that functionality not available?

Possible parameters for an annotation on a class found by hk2 that implements com.sun.enterprise.ee.cms.core.Callback.

1. What GMS notifications should the callback be registered with?
   Current GMS notifications include:

   GMS Membership Notifications: Join, JoinedAndreadReady, PlannedShutdown, FailureSuspected, Failure
   GMS Cluster Event Notification: MessageActionFactory, GroupLeadership, FailureRecovery

   (Implementation detail: append "NotificationActionFactoryImpl" to all of the above when implementing in gms-adapter/GMSService)

2. Which gms-group should the callback be registered with?
   (a) ALL groups
   (b) a specific gms-group

3. Both FailureRecovery and Message notifications are registered by a componentName that is application specific.
   This info would need to be communicated via an annotation or a new method added to interfaces for Message and RecoveryFailure Callbacks.

   See componentName javadoc for GroupManagementService.addActionFactory(String componentName, FailureRecoveryActionFactory)
   and GroupManagementService.addActionFactory(MessageActionFactory, String componentName).

4. Possible application data to be passed to the constructor of the class that implements com.sun.enterprise.ee.cms.core.Callback.
   Minimally, it has been useful to have the memberToken name and gms-group name within Shoal GMS callback handlers.

****

Add diagnostic utility to confirm that multicast is enabled between
a set of machines. This tool diagnoses one issue on why cluster members
may not be seeing each other. The tool only uses java MulticastSocket
and does not use GMS nor its transport. Name for tool and where it will live
are to be determined. Documentation is needed on when and how to use it.

For GMS over Grizzly, this utility probably should be based on a generalized version of

com.sun.enterprise.mgmt.transport.BlockingIOMulticastSender.java.

****

4.5.2. Private Interfaces:

// List private interfaces which are externally observable.

Configuration within Glasfish v3.1 domain.xml element cluster and group-management-service elements.

See specification in http://wiki.glassfish.java.net/attach/V3FunctionalSpecs/gmsconfig_gfv3_1.rtf

New module: cluster/gms-adapter   // depends on shoal-gms-api.jar and shoal-gms-impl.jar

@Service

class GMSAdapterImpl implements GMSAdapter

4.5.3. Deprecated/Removed Interfaces:

// List existing public interfaces which will be deprecated or

// removed by this project.

NONE

4.6. Doc Impact:

// List any Documentation (man pages, manuals, service guides...)

// that will be impacted by this proposal.

Document new diagnosis utility to confirm that multicast is enabled on a subnet.

Assists in diagnosing why dynamic clustering is not working for a set of clustered instances.

When it is obvious that the clustered instances in a glassfish cluster are not seeing each other

(via analysis of GMS views in server log files), this utility should be run on each machine.

This utility assists in diagnosing if multicast is possible among

the machines being used to
host the clustered instances.

4.7. Admin/Config Impact:
// How will this change impact the administration of the product?
// Identify changes to GUIs, CLI, agents, plugins...

Changes are required for Admin GUI configuration of a cluster and group-management-service.
See http://wiki.glassfish.java.net/attach/V3FunctionalSpecs/gmsconfig_gfv3_1.rtf

There is a need for being able to specify generic gms properties on both cluster and group-management-service level.
In Glassfish v2.1, there was no way to specify generic properties at the cluster level, only group-management-service level.
Thus, cluster properties could only be added via asadmin CLI set command in glassfish v2.x.

4.8. HA Impact:
// What new requirements does this proposal place on the High
// Availability or Clustering aspects of the component?
None

HA has requirements that GMS must meet.
Identified requirements are in this document.
There does exist a chance that other requirements are identified during implementation phase
given the relationship between HA and GMS.

4.9. I18N/L10N Impact:
no.

4.10. Packaging, Delivery & Upgrade:

4.10.1. Packaging
// What packages does this proposal impact? How will the packages
// be impacted? Will new IPS/pkg(5) packages need to be created?

Not sure what a package is, but here is how Shoal GMS is integrated into Glassfish v3.1.

Glassfish v3.1 module cluster/gms-adapter loads GMS and joins a glasfish clustered instance to its gms group.

Shoal GMS is a sub project of Glassfish. The shoal gms module is

external source code that has its jar checked into Glassfish v3.1 via glassfish
maven repository.

The Shoal GMS source code exist under a glassfish subproject on java.net.
Sources are located here:can be retrieved here.
svn checkout https://shoal.dev.java.net/svn/shoal/branches/SHOAL_1_1_ABSTRACTING_TRANSPORT --username <java.net.userid>

For the checked out source code, see pom.xml and directory gms, for Shoal GMS module.

Final jar name: shoal-gms-1.5.jar. The updateable jar file is called shoal-gms-1.5-SNAPSHOT.jar.

4.10.2. Delivery
// What impact will this proposal have on product installation?
The module will be installed as part of Glassfish install.

4.10.3. Upgrade and Migration:
// What impact will this proposal have on product upgrade and/or
// migration from prior releases? Enumerate requirements this
// project has on upgrade and migration.

Since clustering was not in Glassfish v3, there is no upgrade from v3 to v3.1.

Upgrade from glassfish v2 clustering applications to glassfish v3.1 is needed.
This will consist of identifying glassfish v2 cluster and group-management-service elements in v2 domain.xml and
mapping to the glassfish v3.1 configuration. The gms configuration document captures what transformations
will need to be made.

4.11. Security Impact:
// How does this proposal interact with security-related APIs
// or interfaces? Does it rely on any Java policy or platform
// user/permissions implication? If the feature exposes any
// new ports, Or any similar communication points which may
// have security implications, note these here.

GMS using Grizzly ussing SSL is not in scope.

4.12. Compatibility Impact

// Incompatible changes to interfaces that others expect

```
        // to be stable may cause other parts of application server or
        // other dependent products to break.

        // Discuss changes to the imported or exported interfaces.
        // Describe how an older version of the interface would
        // be handled.
        Missed GMS Notification of FAILURE when a clustered instance is
restarted quicker than configured GMS heartbeat failure detection
parameters.
          In Glassfish v2.1.1,  the Nodeagent was a GMS Watchdog that
reported to GMS that a clustered instance had
          failed.(Described in http://wiki.glassfish.java.net/attach/
SFv2FunctionalSpecs/gmsWatchdog.doc)
          This was done since the Nodeagent was restarting the instance on
some machine configurations quicker
          than GMS heartbeat failure detection could report the instance
had failed.  By the time GMS was confirming a failure,
          the clustered instance had been restarted (quicker than the
default of 8 seconds it takes for GMS heartbeat failure
          detection to confirm a failed instance.)  For Glassfish v3.1,
the GMS client is required to register
          a JOIN and/or JOINED_AND_READY callback that check for the
existence of the newly introduced subevent REJOIN.
          When this subevent exists on a JOIN or JOINED_AND_READY
notification, it indicates that the instance restarted
          without a GMS notification that the instance had failed. The
REJOIN subevent represents a recent FAILURE
          of the clustered instance that is (re)JOIN the glassfish cluster.
The INFO log event for GMS notification JOIN and JOINED_AND_READY
          will have REJOIN in it, if there is a REJOIN element.

     4.13. Dependencies:
        // List all dependencies that this proposal has on other
        // proposals, components or products.
        // LIST dependency component version requirements here.

        grizzly-framework.jar version 1.19.9-beta2 or higher  (GMS code
will not compile with 1.19.8 or lower)
        grizzly-utils.jar     version 1.19.9-beta2 or higher

        Admin GUI for ability to configure cluster and group-management-
service properties.
        asadmin CLI for setting domain.xml cluster and group-management-
service attribute and properties.

        Depend on symbolic token replacement in domain.xml to set
BIND_INTERFACE_ADDRESS different for each clustered instance.
        ${<cluster-name>-GMS_BIND_INTERFACE_ADDRESS}.

        Depend on existence of "asadmin start-cluster" to inject
```

supplemental GMS behavior to call
GroupManagmentService.announceGroupStartup()

Distributed testing of GMS in GF v3.1 environment relies on remote starting/stopping of cluster and instances.
asadmin start-cluster, stop-cluster, start-instance, stop-instance.

Subevent REJOIN of ADD event should be implemented by time that automatic restart of a failed clustered instance is in GF v3.1.

Developer level testing of a cluster on one machine requires the ability to easily start multiple clustered instances on one machine.
In Glassfish v2.1, creating multiple instances on one machine resulted in each instance using different ports for HTTP, IIOP, ...
CONFIG-05 feature provides this.

4.14. Testing Impact

// How will the new feature(s) introduced by this project be tested?
REJOIN -
A SHOAL GMS developer unit level test is already written that will cause a REJOIN to occur.
The JOIN and JOIN_AND_READY ShoalLogger INFO log event should mention REJOIN subevent when it exists.

Existing scenarios from gf v2.1 gms with kill of a clustered instance will need to change its log analysis to verify REJOIN.
The REJOIN will only be testable in Glassfish v3.1 platform when native OS restart of application server is implemented
and appropriate instructions to activate that property are followed (if there are any)
(REJOIN replaces NodeAgent WATCHDOG from gf v2.1.1. Since NodeAgent is not implemented, NodeAgent WATCHDOG no longer will work in
GF v3.1)

****

New method previousAliveAndReadyMembers().
A new Shoal GMS developer unit level test will be written to verify new method GroupHandle.previousAliveAndReadyMembers().
The test will verify that this method is working properly by starting up a cluster, ensuring it is at steady state
and then verify that this method returns the proper list after GMS client has join or left the cluster.
For Glassfish QE distributed Shoal testing, the scenario needs to verify what this method returns when the
in a steady state glassfish cluster after the following operations cause a change in a running glassfish cluster that has reached steady state.

(1) start instance via "asadmin start-instance)
(2) PLANNED_SHUTDOWN of single clustered instance
(3) FAILURE detection of a failed instance. (4) restart of an instance in glassfish v3.1 (via it being registered as a native OS service)

Note: no testable constraints exist for this method during cluster startup (asadmin CLI start-cluster),
during cluster shutdown (asadmin CLI stop-cluster), when multiple events (starts/stops) happen exactly at same time.

****

// Do tests exist from prior releases (e.g. v2) that can be reused?
Yes. Shoal QE Scenario test scenarios can be reused. Adjustments may need to be made to
accommodate differences between v2 and v3.1. (i.e. NodeAgent not existing in v3.1, potentially
minor changes to asadmin CLI to manage glassfish cluster starting/stopping, ...)
These changes will amount to fairly small changes to scripts that run scenarios and
changes to what log events to look for in certain situations. However, the testing methodology
used to test in v2 will work equally well in v3.1.

// Will new tests need to be written? Can they be automated?
New tests to be written were listed above and initial thoughts on testing were mentioned.
It should be possible to automate running the tests but manual investigation of log files of failures
will be needed.

5. Reference Documents:
// List of related documents, if any (BugID's, RFP's, papers).
// Explain how/where to obtain the documents, and what each
// contains, not just their titles.

See GMS configuration in Glassfish v3.1: http://wiki.glassfish.java.net/PageInfo.jsp?page=GlassFishv3.1GMS/gmsconfig_gfv3_1.rtf

Shoal GMS over Grizzly distributed unit QE testing (by Kazem)
http://wikihome.sfbay.sun.com/glassfish/Wiki.jsp?page=ShoalGrizzlyTestScenarios

Following page contains Milestone functionalities.

Glassfish v3.1 GMS subproject page: http://wiki.glassfish.java.net/Wiki.jsp?page=GlassFishv3.1GMS

Reference GMS Configuration in Glassfish v3.1 document here. Describes GMS configuration in domain.xml.

Stop-gap clustering distributed test harness (from Steve D.)
% export CVSROOT=:pserver:<sunLDAPID>@redcvs.red.iplanet.com:/m/jws
% cvs co appserver-sqe/ee/gms/sm
See appserver-sqe/ee/gms/sm/README.txt

Glassfish v2.1.1 functionality not being implemented in Glassfish v3.1 since Nodeagent is not part of GF v3.1.
GMS Watchdog functionality ensured that a clustered instance that is quickly restarted (less than 8 secs) is
still reported as a GMS FAILURE. gf v2.1.1 details described in
See: http://wiki.glassfish.java.net/attach/SFv2FunctionalSpecs/gmsWatchdog.doc

6. Schedule:
6.1. Projected Availability:
* Initially integrated (may not be feature complete) M2
Before M2. Hopefully before end of May. Stop-gap clustering test harness enable earlier testing.
Being held up by task to only load shoal-1.5.jar
if cluster gms-enabled is true.

* Feature complete (ready for handoff to QA) M4

* At production quality level: Final Release

M

Relationships

clustered instance belongs to one glassfish cluster.
standalone instance belongs to a gms group. *new* design on how this appears in GMS config outstanding.
DAS belongs to 0 to many glassfish clusters.

Given above relationships, original implementation of GMSAdapterImpl (formerly called GMSService)
was one to one relationship with GMS group. Is it possible to have more than one

GMSAdapter, one per glassfish cluster that DAS belongs to.

GMS client lifecycle
- join   called once when GF app server instance joins GMS group

- reportJoinedAndReady -   should be called when EventTypes.SERVER_READY occurs. (if that is when server is ready
                             for incoming client traffic.)

- shutdown   called when receive EventTypes.SERVER_SHUTDOWN occurs.

TBD: Access to GMS in DAS with more than one cluster.