FD – External Component Register

Content

History

Version	Date	Adjustments
PA1	2007-08-28	First version of the document. Original document was found in MAS. (EMAHAGL)
PA2	2007-09-26	Updated with service status functionality (EMAHAGL)

1 Introduction

The External Component Register (ECR) handles registration and location of services in the Messaging Component Register (MCR).

For the specification of the ECR, see [1].

2 Function Structure

2.1 Overview

An overview of the component is shown in Figure 1.

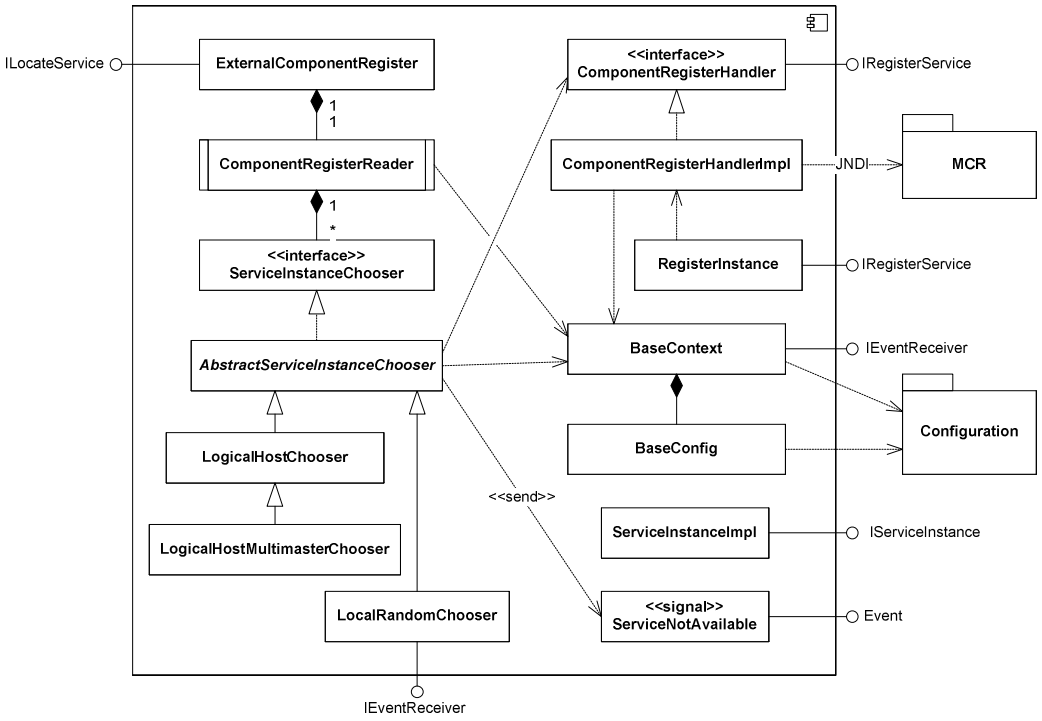

Figure 1 System overview of the ECR

The ExternalComponentRegister class implements the ILocateService interface and is used for retrieving service instances and reporting problems with service instances.

The ComponentRegisterHandlerImpl class manages all communication with MCR. It implements the IRegisterService interface which is used for registering and unregistering service instances in MCR.

The ComponentRegisterHandler is an internal interface mainly used for testing the ExternalComponentRegister and ServiceInstanceChooser implementations.

The ComponentRegisterReader is an active object which at regular intervals polls MCR for available service instances. It is also responsible for delegating service location requests to the correct ServiceInstanceChooser, depending on the service name.

The ServiceInstanceImpl class implements the IServiceInstance interface and contains the properties for each service instance. It also contains a status variable if the service is considered to be up or down.

The BaseContext class is a common context class used by most of the other classes. It maintains the current configuration among other things. It also implements the IEventReceiver interface and receives the ConfigurationChanged event.

The BaseConfig class parses the configuration to a suitable format.

The ServiceInstanceChooser interface defines methods common for all service instance choosing algorithms.

The AbstractServiceInstanceChooser is a common abstract base class for other service instance choosing classes.

The LocalRandomChooser implements the MER algorithm selection of service instances (see [2]).

The LogicalHostChooser implements a logical host selection of service instances (see [3]).

The LogicalHostMultimasterChooser implements a multimaster selection of service instances, i.e. a deployment without consumers (see [3]). In the absence of such a deployment, the LogicalHostMultimasterChooser works as a LogicalHostChooser.

The RegisterInstance is a standalone class providing a shell command interface for registering and unregistering service instances in MCR.

2.2 ExternalComponentRegister

The ExternalComponentRegister delegates the service location calls and service error reports to the ComponentRegisterReader. It implements the ILocateService interface, supplying the following methods:

```
IServiceInstance locateService(String serviceName)
     throws NoServiceFoundException;
IServiceInstance locateService(String serviceName, String hostName)
     throws NoServiceFoundException;
IServiceInstance getAnotherService(IServiceInstance service)
     throws NoServiceFoundException;
void reportServiceError(IServiceInstance service);
List<IServiceInstance> getServiceInstances(String serviceName)
     throws NoServiceFoundException
```

2.3 ComponentRegisterReader

The ComponentRegisterReader delegates service location calls and error reports to ServiceInstanceChoosers, depending on service name. It is an active object which at regular intervals calls the lookup method on all ServiceInstanceChoosers, thereby updating the service instance caches.

The mapping between service name and the name of the ServiceInstanceChooser implementation is received from the BaseContext as a Map<String, String>.

2.4 ServiceInstanceChooser

The ServiceInstanceChooser is an internal interface for all service instance selection algorithms. Each ServiceInstanceChooser corresponds to a service name. The interface contains the following methods:

```
IServiceInstance locateService() throws NoServiceFoundException;
IServiceInstance locateService(String hostName)
     throws NoServiceFoundException;
IServiceInstance getAnotherInstance(IServiceInstance serviceInstance)
     throws NoServiceFoundException;
void reportServiceError(IServiceInstance serviceInstance);
List<IServiceInstance> getServiceInstanceList();
```

```
void lookup();
```

The five first methods are just reflections of the ILocateService interface. The lookup method is used to make the ServiceInstanceChooser renew its service instance cache.

2.5 AbstractServiceInstanceChooser

The AbstractServiceInstanceChooser class contains common functionality for ServiceInstanceChoosers, e.g. the currently selected service instance, configured instances, if MCR is overridden or not and reporting of services which are no longer available (by sending the ServiceNotAvailable event).

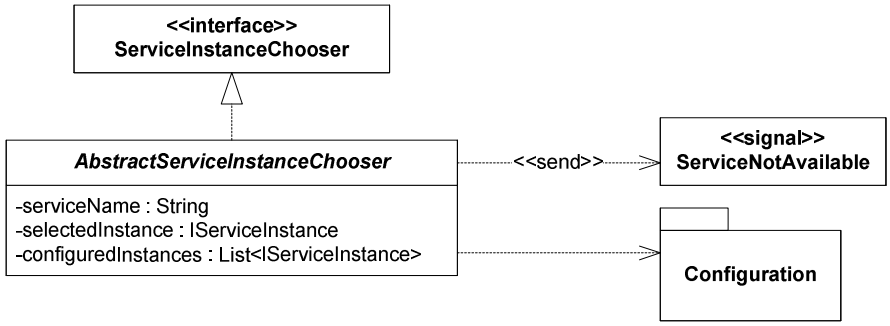

Figure 2 AbstractServiceInstanceChooser

2.6 LocalRandomChooser

The LocalRandomChooser implements the MER algorithm for selecting service instances (see [2]).

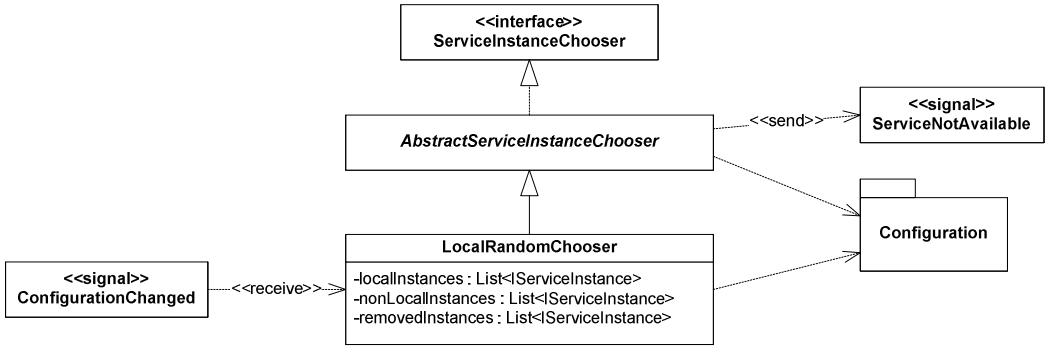

Figure 3 LocalRandomChooser

It is possible to configure which logical zone and IP subnet the LocalRandomChoosers belong to. These are used for deciding if a service instance should be considered close or not according to the MER algorithm. The LocalRandomChooser listens for ConfigurationChanged events to be able to update these configured properties.

2.7 LogicalHostChooser and LogicalHostMultimasterChooser

The LogicalHostChooser implements the logical host selection algorithm (see [3]). The LogicalHostMultimasterChooser extends the LogicalHostChooser with multimaster selection functionality for pure multimaster environments (see [3]).

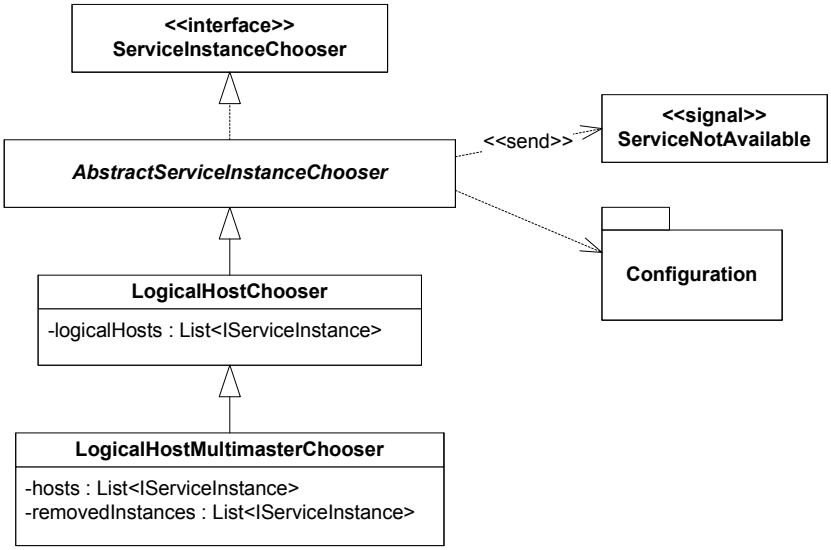

Figure 4 LogicalHostChooser and LogicalHostMultimasterChooser

2.8 ComponentRegisterHandlerImpl

The ComponentRegisterHandlerImpl class is responsible for the communication with MCR, i.e. searching, registering and unregistering service instances. It uses JNDI for this. The environment parameters for the JNDI DirContext creation are retrieved from the BaseContext as a Hashtable<String, String>.

It implements the IRegisterService interface and is thereby also the factory for IServiceInstance objects.

2.9 BaseContext

The BaseContext class is common to all classes in the ECR module. It contains common resources used by the other classes, e.g. event dispatcher and configuration. It creates the BaseConfig object and keeps the configuration up to date by listening to the ConfigurationChanged event.

The BaseContext is also used for rescheduling the periodic lookup when the ConfigurationChanged event has been received.

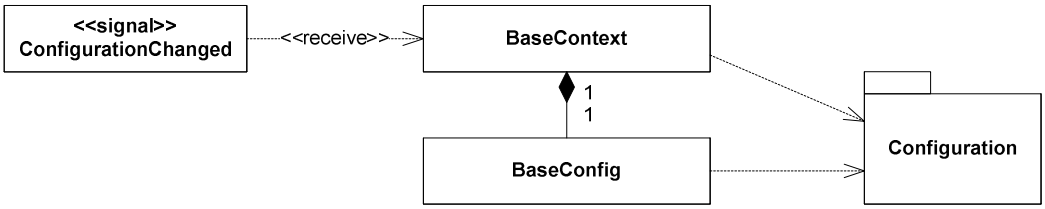

Figure 5 BaseContext and BaseConfig

2.10 RegisterInstance

The RegisterInstance class provides a shell command line interface for registering and unregistering service instances in MCR. Usage:

```
RegisterInstance {register|unregister} servicename attribute=value...
```

The RegisterInstance uses the Spring framework to retrieve a class implementing IRegisterService. The Spring configuration is specified in RegisterInstanceConfig.xml.

3 Function Behavior

3.1 ComponentRegisterReader

3.1.1 Startup

The startup behavior for the ComponentRegisterReader is described in Figure 6.

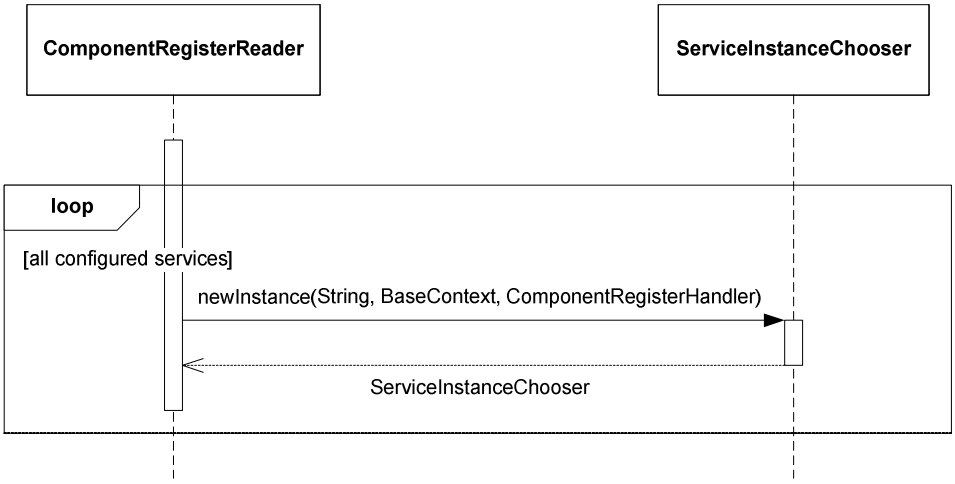

Figure 6 Startup

For each service found in the configuration, the ComponentRegisterReader creates a new instance of a ServiceInstanceChooser providing the service instance selection algorithm for the configured service. The mapping between service name and ServiceInstanceChooser implementation is received by dependency

injection. If no mapping exists or the class cannot be instantiated, the LocalRandomChooser implementation is selected.

3.1.2 Update

The update behavior is described in Figure 7.

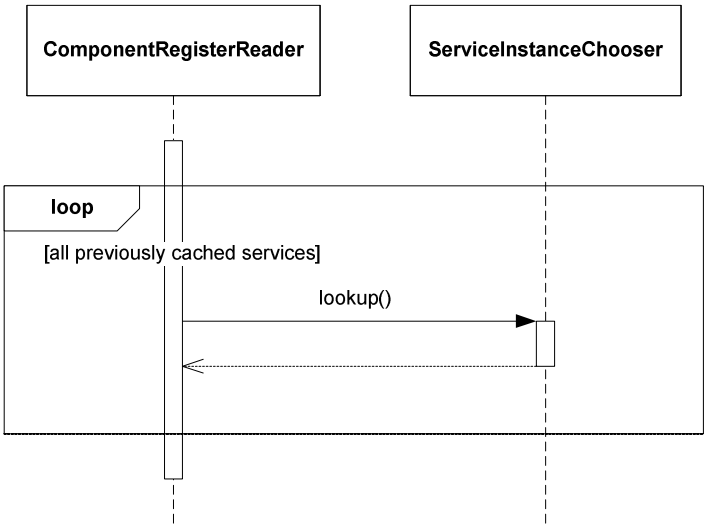

Figure 7 Update

At regular intervals, for each service previously cached, the ComponentRegisterReader calls the lookup method on the instance of the ServiceInstanceChooser providing the service instance selection algorithm for the configured service. This is handled by a ScheduledExecutorService retrieved from the BaseContext (see 3.6).

3.1.3 Unknown service

If an instance for a currently unknown service is requested, a new LocalRandomChooser instance for that service is created and added to the services handled by the ComponentRegisterReader. The service instances for this service will subsequently be updated at regular intervals, as described in section 3.1.2.

3.2 LocalRandomChooser

3.2.1 Lookup

Each time the LocalRandomChooser updates its internal cache of service instances it checks to see if a service instance has been removed since the previous search. This is done by comparing the search result with the instances in the lists of local, non-local and removed instances. For each service instance removed a ServiceNotAvailable event is sent.

It is also checked if the currently selected service instance should be changed. This is only done if the instance is no longer available (no longer found in MCR) or

the priority of the currently selected instance is lower than another instance that could be selected, e.g. if currently a non-local or configured host is selected but the latest search returned a local host.

3.2.2 LocateService

When locateService() is called, the currently selected instance is returned. If no current instance is selected NoServiceFoundException is thrown.

When locateService(String hostName) is called, the lists of local, non-local and configured instances are searched for a service instance having the supplied hostname. The first such service instance found is returned. If no instance is found NoServiceFoundException is thrown.

3.2.3 GetAnotherInstance

When a request for another instance is sent to the LocalRandomChooser, each of the lists of local, non-local and configured instances are searched for a service instance different from the submitted service instance. If no service instance is found, a NoServiceFoundException is thrown.

3.2.4 ReportServiceError

When a service instance is reported to the LocalRandomChooser, it is removed from any of the lists of local, non-local or configured instances and added to the removed instances list. It is also checked if it was the currently selected service instance that was reported, in which case it is replaced.

If the current instance is removed and no new instance could be selected, a NoServiceFoundException will be thrown for a configurable amount of time. After the time has expired, a new lookup is performed.

3.3 LogicalHostChooser

3.3.1 Lookup

Each time the LogicalHostChooser updates its internal cache of logical service instances (there should be only one instance), it checks to see if a service instance has been removed since the previous search. For each service instance removed a ServiceNotAvailable event is sent.

It is also checked if the currently selected service instance should be changed. This is only done if the instance is no longer available (no longer found in MCR).

If no logical hosts are found, a configured host is selected. If no configured host is found, no host will be selected.

3.3.2 LocateService

When locateService() is called, the currently selected instance is returned. If no current instance is selected NoServiceFoundException is thrown.

When locateService(String hostName) is called, the lists of logical and configured instances are searched for a service instance having the supplied hostname. The

first such service instance found is returned. If no instance is found NoServiceFoundException is thrown.

The search is performed by traversing the list and comparing the service instance hostname with the supplied hostname.

3.3.3 GetAnotherInstance

When a request for another instance is sent to the LogicalHostChooser, the currently selected logical service instance is returned. The LogicalHostChooser should always return the current logical host, since this could represent a load balancer hiding many hosts providing the service.

3.3.4 ReportServiceError

When a service instance is reported to the LogicalHostChooser nothing happens. The LogicalHostChooser should always return the current logical host, since this could represent a load balancer hiding many hosts providing the service.

3.4 LogicalHostMultimasterChooser

The LogicalHostMultimasterChooser is only different from the LogicalHostChooser when it detects a pure multimaster environment, i.e. all found service instances has a replication id less than or equal to 255.

3.4.1 Lookup

Each time the LogicalHostMultimasterChooser updates its internal cache of service instances, it checks to see if a service instance has been removed since the previous search. For each service instance removed a ServiceNotAvailable event is sent.

It is also checked if the currently selected service instance should be changed. This is only done if the instance is no longer available (no longer found in MCR).

The list of multimaster hosts is sorted in ascending replication id. The Comparator used is the ReplicationIdComparator, which only compares the service instance replication id's. An instance without replication id is considered greater than an instance with replication id. **Note:** this Comparator imposes orderings which are inconsistent with equals, i.e. two service instances with the same replication id but different host name, will return 0 when compared.

3.4.2 LocateService

When locateService() is called, the currently selected instance is returned. If no current instance is selected NoServiceFoundException is thrown.

When locateService(String hostName) is called and there is a multimaster environment, the list of multimaster instances are searched for a service instance having the supplied hostname. The first such service instance found is returned. If no instance is found the lists of logical and configured instances are searched. If still no instance is found, NoServiceFoundException is thrown.

The search is performed by traversing the lists and comparing the service instance hostname with the supplied hostname.

3.4.3 GetAnotherInstance

When a request for another instance is sent to the LogicalHostMultimasterChooser and there is a multimaster environment the lists of multimaster and configured instances are searched for a service instance different from the submitted service instance. If no service instance is found, a NoServiceFoundException is thrown.

3.4.4 ReportServiceError

When a service instance is reported to the LogicalHostMultimasterChooser and there is a multimaster environment, it is removed from any of the lists of multimaster or configured instances and added to the removed instances list. It is also checked if it was the currently selected service instance that was reported, in which case it is replaced.

If the current instance is removed and no new instance could be selected, a NoServiceFoundException will be thrown for a configurable amount of time. After the time has expired, a new lookup is performed.

3.5 ComponentRegisterHandlerImpl

3.5.1 Search

The ComponentRegisterHandler searches MCR for service instances providing a certain service. The service name used in the search is retrieved from the configuration attribute servicename, using the submitted service name as key. If no servicename attribute exists, the submitted service name is used as it is.

3.5.2 Timeouts and retries

JNDI does not support client timeout for its methods. The timeouts in JNDI are on the server side. This means that if the MCR is not able to handle a request, the call could block longer than the configured timeout. To be able to handle such problems with the MCR host, the TimeoutRetrier class from the com.mobeon.masp.util package is used. It will execute a Callable in a separate thread, interrupting the thread if the timeout expires. The TimeoutRetrier will also retry executing the Callable in case of certain failures (indicated by throwing a RetryException encapsulating the real exception) a configured number of times during a configured time period. The timeout, try limit and try time limit are all read from configuration.

The search and modify requests throw RetryException when catching a JNDI CommunicationException. In those cases the MCR host is also logged as unavailable using the HostedServiceLogger (see [4]).

3.6 ConfigurationChanged

Almost all classes use the BaseConfig class to retrieve configuration parameters. The BaseConfig is retrieved from the BaseContext. BaseContext listens to ConfigurationChanged events. When received, the BaseContext tries to create a new BaseConfig object with the new configuration. If the creation succeeds, the BaseConfig object is replaced, otherwise an error log is created and the previous configuration is used.

To be threadsafe, the BaseConfig and IConfiguration getters (and modifiers) are synchronized.

The BaseContext also controls the scheduling of the ComponentRegisterReader. If the period is changed by the ConfigurationChanged event, the task is rescheduled with the new periodicity.

Implementations of ServiceInstanceChooser could have their own configurations. In that case they are located in the configuration section algorithms, having the ServiceInstanceChooser class name in lowercase as the configuration group name. To support reloadconfig, such implementations must also listen for the ConfigurationChanged event and reread its configuration upon receiving the event.

4 References

[1] FS External Component Register
6/FS-CRH 109 581-1 Uen

[2] MCR – Developer's Guide
3/1551-CRH 109 581/1

[3] MUR Developers Guide
3/1551-CRH 109 086

[4] FD Log Manager
2/FS-CRH 109 581-1 Uen

5 Terminology

ECR	External Component Register
JNDI	Java Naming and Directory Interface
LDAP	Lightweight Directory Access Protocol
MCR	Messaging Component Register
MER	Messaging Event Repository

6 Appendix: 3PP

3PPName / Freeware Name	Version of the product/ freeware	Company	Used for	Delivered with the component	ECCN US/EU	Product No. and R-state
Spring Framework	1.2.7	Spring	Setting class dependencies for command line interface	Yes	EAR99	SWF0060R1A