Functional Specification for DCR Plug-in Support

Author(s): joel.binnquist.xc@ericsson.com

Version: 1.3

Version	Date	Comment
0.1	2009-01-20	First version
1.0	2009-04-02	Updated after review. - Removed support for annotations - Removed support for runtime compilation
1.1	2009-04-02	Updated API section.
1.2	2009-04-03	Removed paragraph mentioned java source code.
1.3	2009-04-02	Added link to FSD for number normalization.

1 Introduction

The Data Centric Rules (DCR) is a part of the Converged Load Balancer making it possible to configure how the key, used as input to the consistent hash, is extracted from a request.

The rules for a specific CLB configuration can currently be specified using a proprietary language in XML. However, this language is somewhat limited, and moreover, it is hard to understand.

The Presence and Group Management application (PGM) has put on a requirement that Sailfin shall support the ability to specify DCR rules where one can write a condition on a specific SIP method. Something like:

if *req.method equals REGISTER* ***then***
  extract key from req.uri
else
  extract key from req.myheader

Today one cannot specify a comparison condition on a request method; the only entity that one can express a comparison condition on is the session case. This could be solved by enhancing the existing language.

While analyzing this it was suggested that applications should be able to configure DCR via plug-ins (written in Java) rather than enhancing the proprietary language. Thus when the CLB needs to extract the hash key from a request it calls the currently installed plug-in.

The advantages with a plug-in solution would be:

The language is well-known, which makes it easier for the application developer, who shall write the rules.

The solution will be easier to maintain for the Sailfin development organization compared to a proprietary XML based language

The solution will be much more flexible for the applications; the applications will be able to use the full power of the Java language when expressing the rules.

It will be possible for the applications to verify and debug the rules using standard tools; today this is not possible.

There will be no need to spend design hours for enhancement of a proprietary the language in the future.

One important constraint is that Sailfin supports dynamic reconfiguration of the rules, which requires that the plug-in classes can be unloaded or reloaded.

It shall be possible to install/download a new plug-in via the CLI and Admin GUI without having to copy it to the file system of Application Server (in a manner similar to deploying applications).

The old solution with the XML-based language should be deprecated or discontinued, to minimize future costs for maintenance.

2 Function

2.1 Overview

2.1.1 Configuration

The Data Centric Rules of the Converged Load Balancer will be configurable using an implementation of the interface *org.glassfish.comms.api.datacentric.DcrPlugin*, which is supplied as a precompiled Java class packaged in a jar-file or. The application server determines how to interpret the file based on the file extension.

The user provides the file using the existing interfaces for DCR configuration (Admin GUI and CLI). The execution of a “set DCR file” command (GUI or CLI) causes the admin framework to upload the file to an internal directory in the application server. (This mechanism is already implemented for the DCR XML.).

The admin framework generates an admin event into the CLB where the file name is supplied as an argument (note that the admin framework does not care about what kind of file this is, it is up to the CLB to decide how to process the file). The CLB then selects how to process the file based on file extension (case-insensitive):

- If it is a .xml file, then it is processed as a DCR XML file (the existing solution).
- If it is a .jar file, it is expected to be a jar-file containing a compiled implementation of the plugin interface (and perhaps other application specific helper classes) and where the manifest specifies the class name of this class. The CLB then loads the plugin class and creates on instance.

2.1.2 Message Routing

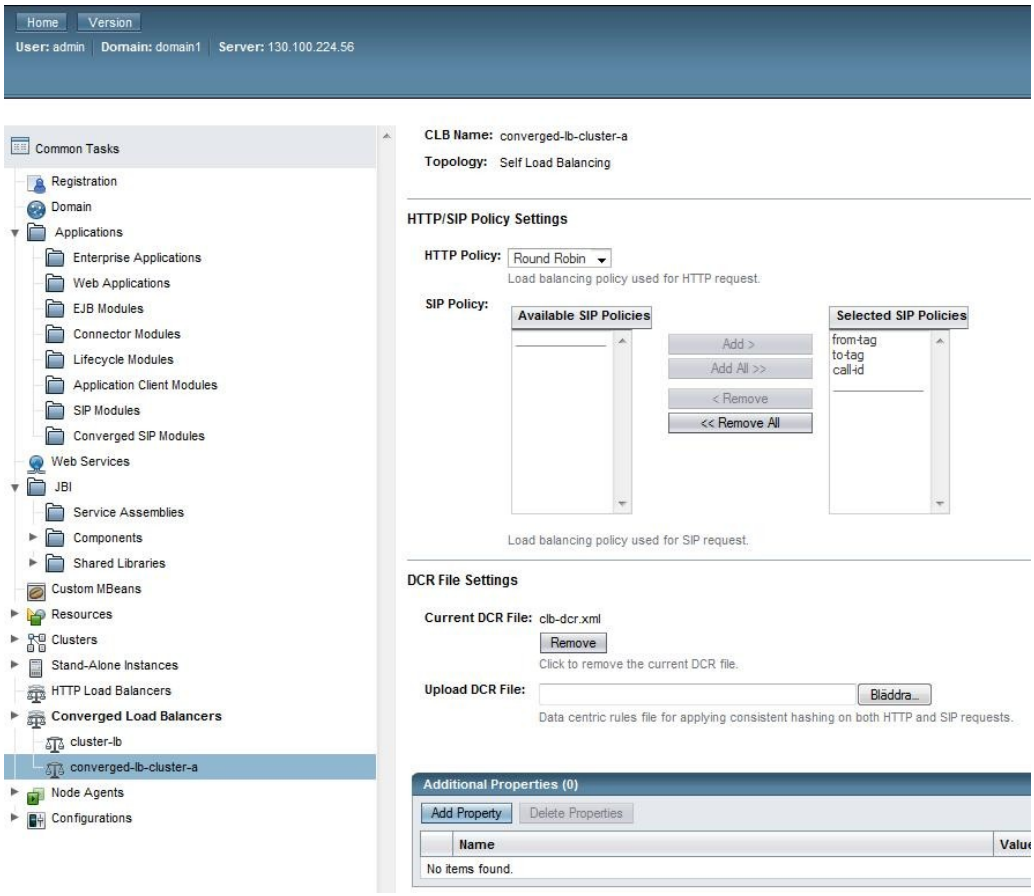
When routing a request, the CLB calls the instantiated plug-in to extract the key from initial[1] SIP and HTTP requests. The CLB uses the result returned from the plug-in, to lookup the serving instance in the consistent hash.

Note, if required the plug-in is responsible for resolving Tel-URI:s, normalize numbers and canonicalize URI:s (see Functional Specification for Number Normalization and URICanonicalization). The API:s required for this are provided as arguments on the method call to the plug-in (see 2.4.1).

[1] A request is initial if it is not part of an ongoing session/dialog.

2.2 Configuration Interface

The file is supplied either via the CLI commands: *create-converged-lb*, *create-converged-lb-config* or *set-dcr-file* or via Converged Load Balancer configuration interface in the Admin GUI:



2.3 Configuration

2.4 API

2.4.1 DcrPlugin

The org.glassfish.comms.api.datacentric.DcrPlugin specifies two methods:

- String getKey(javax.servlet.sip.SipServletRequest request, javax.servlet.sip.SipFactory sipFactory, org.glassfish.comms.api.uriutils.UriTools uriTools, org.glassfish.comms.api.telurl.TelUrlResolver telUrlResolver)
- String getKey(javax.servlet.http.HttpServletRequest request, javax.servlet.sip.SipFactory sipFactory, org.glassfish.comms.api.uriutils.UriTools uriTools, org.glassfish.comms.api.telurl.TelUrlResolver telUrlResolver)

The implementor shall implement these to extract the key to be used for lookup in the consistent hash in the CLB, for initial requests. (If *null* is returned the CLB will fall back to the default behavior, i.e. create a key from the call-id and from-tag of the request).

In order to facilitate implementation of the DcrPlugin the following entities are provided as method arguments:

- TelUrlResolver (to be able to resolve Tel-URI:s)
- UriTools (to be able to normalize and canonicalize URI:s)
- SipFactory (to be able to convert header values to URI:s and extract the various components from such a URI)

2.4.2 JAR-file

The jar-file shall contain the plug-in-class and possibly other classes as needed by the plug-in-class. Note, that the class is loaded by a separate class loader which has the container classloader as parent, thus classes that are not part of the container, needed by the plug-in, must be provided in the jar-file.

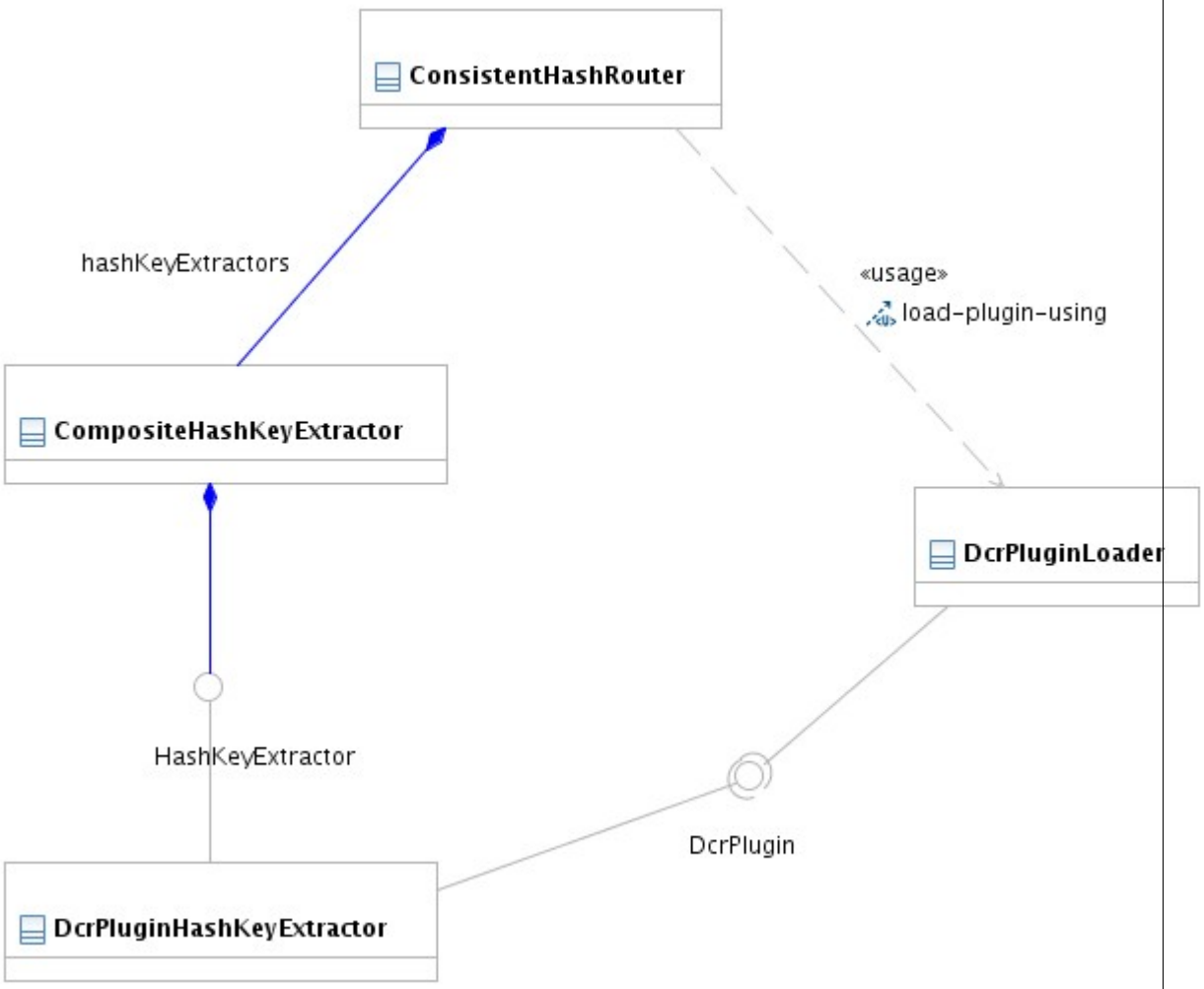
The manifest shall specify the plug-in-class using the attribute *Dcr-Plugin-Class*, as below:

```
Manifest-Version: 1.0
Dcr-Plugin-Class: com.example.dcrplugin.ExampleDcrPlugin
```

2.5 Constraints

3 Design Overview

3.1 Class Diagram



Each time the configuration changes and the DCR-file changes the *ConsistentHashRouter* is triggered to re-configure.

The *ConsistentHashRouter* reads the name of the DCR-file and instantiates a *DcrPlugin* using the *DcrPluginLoader*. The *ConsistentHashRouter* pushes the instantiated *DcrPlugin* onto the set of *hashKeyExtractors*)

The *DcrPluginLoader* handle loading of the DCR-plugin which is a jar-fil.

4 Quality and Availability

N/A

5 Performance

-

6 Management and Monitoring

6.1 Formal Interfaces

See *API Interfaces*.

7 Packaging, Files, and Location

8 Documentation Requirements

A user guide how to write a handler class, declare mappings and install the plug-ins is required.

9 Open Issues

None.

10 API Interfaces

10.1 DcrPlugin

```
package org.glassfish.comms.api.datacentric;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.sip.SipServletRequest;
import javax.servlet.sip.SipFactory;

import org.glassfish.comms.api.datacentric.DcrPlugin;
import org.glassfish.comms.api.telurl.TelUrlResolver;
import org.glassfish.comms.api.uriutils.UriTools;

public interface DcrPlugin {
    String getKey(SipServletRequest request, SipFactory sipFactory,
UriTools uriTools, TelUrlResolver telUrlResolver);
    String getKey(HttpServletRequest request, SipFactory sipFactory,
UriTools uriTools, TelUrlResolver telUrlResolver);
}
```

11 Example Plug-in

The following is an example of a plug-in implementation:

```
package com.example.dcrplugin;

import org.glassfish.comms.api.datacentric.DcrPlugin;
import org.glassfish.comms.api.telurl.TelUrlResolver;
import org.glassfish.comms.api.telurl.TelUrlResolverException;
import org.glassfish.comms.api.uriutils.UriTools;

import java.io.IOException;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.sip.ServletParseException;
import javax.servlet.sip.SipFactory;
import javax.servlet.sip.SipServletRequest;
import javax.servlet.sip.SipURI;
import javax.servlet.sip.URI;


public class ExampleDcrPlugin implements DcrPlugin {
    @Override
    public String getKey(SipServletRequest request, SipFactory sipFactory,
UriTools uriTools, TelUrlResolver telUrlResolver) {
        URI uri;

        if (request.getMethod().toUpperCase().equals("REGISTER")) {
            String header = request.getHeader("My-Header");

            try {
                uri = sipFactory.createURI(header);
            } catch (ServletParseException e) {
                uri = request.getRequestURI();
            }
```

```
            } else {
                uri = request.getRequestURI();
            }

            URI canonicalizedUri = uriTools.canonicalize(uri);

            if (canonicalizedUri.isSipURI()) {
                SipURI resolvedUri;

                try {
                    resolvedUri = telUrlResolver.lookupSipURI(canonicalizedUri);
                } catch (IOException e) {
                    return null;
                } catch (TelUrlResolverException e) {
                    return null;
                }

                return resolvedUri.getUser() + "@" + resolvedUri.getHost();
            } else {
                return canonicalizedUri.toString();
            }
        }

        @Override
        public String getKey(HttpServletRequest request, SipFactory sipFactory,
    UriTools uriTools, TelUrlResolver telUrlResolver) {
            return request.getRequestURL().toString();
        }
    }
```