Functional Specification for Number Normalization and URI Canonicalization

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

Version: 1.4

Version	Date	Comment
1.0	2008-09-12	First version
1.1	2008-09-16	Updated after review comments.
1.2	2008-09-18	Updated after decision about how to configure mappings.
1.3	2008-09-18	Added information about how to access the API.
1.4	2008-09-18	Added information about directories used in clustered environment.

1 Introduction

This document describes the solution for handling number normalization and URI aliasing in Sailfin.

Number Normalization

Number normalization is used to translate a local number, with a phone context, specified in a Tel-URI according to RFC3966. Such a local number can be specified either as a pure Tel-URI, e.g., *tel:81234567;phone-context=+46*, or as part of a SIP-URI, e.g., sip: *81234567;phone-context=+46@example.com;user=phone*.

URI Canonicalization

URI canonicalization is a function aimed find the canonical representation of a URI alias.

URI aliasing is a (non-standardized) concept where one URI can have several aliases. For example, alice@alpha.com and AliceAtHome@alpha.com could refer to the same user. URI aliasing could also be used to translate telephone numbers not specified according to RFC3966, e.g. the local number *sip:00468123456@example.com* (note that it does not contain *user=phone*, thus the user is not considered being a telephone number according to RFC3966; as a consequence number normalization is not applicable) to the global number *sip:+468123456@example.com;user=phone*.

2 Function

2.1 Configuration

Normalization/Canonicalization procedures are implemented by the application or the deployer in the form of POJO:s. These are plugged into the application server. This is done by apcking them into one or more jar files and dropping the jar files into the <sailfin>/domains/<domain>/lib (<sailfin>/domains/<domain>/config/<cluster-

config>/lib may be used for clusters) and adding the path into the classpath suffix of the application server.
Moreover the deployer must map the plugin-class to a regular expression pattern that selects the specified plugin. This mapping is specified as a property in the sip-service. The properties shall follow the following syntax to be recognized as mappings:
URI-Alias handler: `<property name="uri-alias-mapping<suffix> value="<handler-class-name>;<regular-expression[1]>" />`
Phone context handler: `<property name="phone-context-mapping<suffix> value="<handler-class-name>;<regular-expression>" />`

suffix: this can be an arbitrary string, but it determines the order in which patterns are checked.
handler-class-name: the fully qualified name of the plugin-class.
regular-expression: a regular expression[2] which is checked and if it is matched selects the handler class.

Example:

```
<sip-service>
  :
  <property name="phone-context-pattern1"
            value="com.myapp.MyOperatorSpecificNumberNormalizer;
                   \+46702.*"/>
  <property name="phone-context-pattern2"
            value="com.myapp.MyGeneralSwedishNumberNormalizer;\+46.*"/>
  <property name="uri-alias-pattern1"
            value="com.myapp.MyHomeUriCanonicalizer;.*Home.*"/>
  <property name="uri-alias-pattern2"
            value="com.myapp.MyGeneralUriCanonicalizer;.*"/>
</sip-service>
```

The application server selects the appropriate handler class by applying the regular expressions (in the order specified by the *suffix* described above) on the incoming *phone-context* (number normalization) or URI (URI canonicalization). The application server then invokes a method on the handler class to do the translation.

Mappings can be changed at any time without the need for restart, but when adding/replacing/removing the jar-files, the application server must be restarted in order for it to take effect.

2.2 Converged Load Balancer

[1] See http://java.sun.com/j2se/1.5.0/docs/api/java/util/regex/Pattern.html
[2] See http://java.sun.com/j2se/1.5.0/docs/api/java/util/regex/Pattern.html

The application server performs URI canonicalization and number normalization in the data centric hash key extraction in of the Converged Load Balancer.
When extracting the hash key from a URI, it is canonicalized and possibly normalized (in case it is a Tel-URI).

2.3 DNS resolver

The application server performs number normalization (only), when resolving the IP address of a request to be sent.

2.4 API

The translation functions are exposed in an API.

The number normalization is available in the org.glassfish.comms.api.telurl.TelUrlResolver which can be accessed as a servlet context attribute named with the name as specified in TelUrlResolver.CONTEXT_ATTRIBUTE_NAME.

The URI canonicalization is available in the org.glassfish.comms.api.uriutils.UriUtil which can be accessed as a servlet context attribute with the name as specified in UriUtil. CONTEXT_ATTRIBUTE_NAME.

2.5 Constraints

Since number normalization and canonicalization is done in the Converged Load Balancer, which is long before the serving application(s) has been determined, it is not possible to specify translation rules per application but must be common for the entire application server.

However, the rules governing the translation may differ from application to application. But since rules must be configured for the application server as a whole, it may be required that rules are shared or modified/combined when several applications are deployed on the same application server, since different applications may have different rule implementations for the same phone-context/alias pattern.

The application server assumes that the result of a DNS lookup is always a global number or a canonical URI. Thus it will not do canonicalization/number normalization on the result from such a lookup.

3 Design Overview

3.1 Class Diagram

The entities that are encircled by the dashed line are those are new or being modified when implementing this function.

At startup the Translator reads the configuration and instantiates one object of each of the specified handler classes. Each handler object is registered into a list (there is one list for number normalization handlers and one for URI canonicalization handlers) together with the regular expression selecting it.

The Translator also adds itself as a listener for configuration changes.

When the CLB processes extracts the hash key using DCR it uses the API:s: *TelUrlResolver* and *UriUtil* (new) in order to canonicalize URI:s and resolve Tel-URI:s. These API:s are also available to the application.

When a request is sent the ResolverManager will do normalization via the DnsResolver.

4 Quality and Availability

N/A

5 Performance

It could be good to measure the performance impact when having long lists of mappings.

6 Management and Monitoring

6.1 Formal Interfaces

See *Packaging, Files, and Location*.

7 Packaging, Files, and Location

The plug-ins are packed into a .jar file together with the property files declaring the mappings. The .jar file is placed into *lib* directory of the application server and is then added to the *Classpath Sufffix* of the application server, for example via the web GUI:

The mappings are specified as properties in the sip-service, for example via the web GUI:

8 Documentation Requirements

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

9 Open Issues

None.