Cache implementation in OpenNCP

This document intends to serve as a basis for some brainstorming about the integration of the caching mechanism in OpenNCP (in the scope of implementing the second part of the SMP workflow – download and usage of configurations). This integration must take into account the current architecture refactoring being carried on by OpenNCP to make it conformant with the epSOS specifications and also the need to maintain both TSL and SMP-based Central Services during some time.

Document history:

Author	Version	Date	Comments
João Cunha	0.1	12/02/2016	Initial draft

Analysis

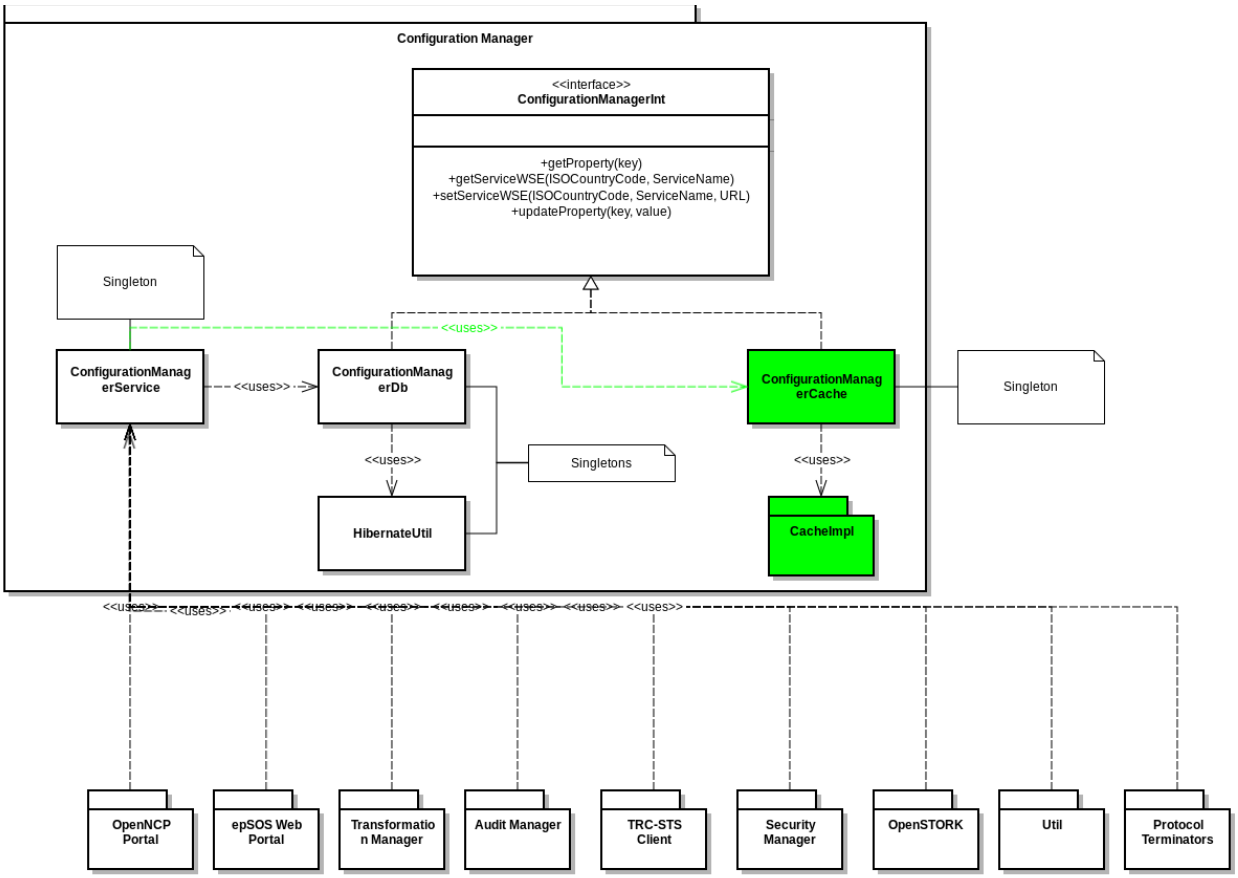

Figure 1: ConfigurationManager architecture

Figure 1 depicts:

1) Architecture of ConfigurationManager component;
2) OpenNCP components depending on ConfigurationManager;
3) Proposal for introducing the new caching layer (marked with green color).

ConfigurationManager component acts as a set of Singleton classes that allow read/write access to the properties stored in the epsos_configuration database. Other OpenNCP components access this component by means of a single entry point: ConfigurationManagerService class. Following is an

example of how Protocol Terminators access ConfigurationManager:

```
String location = configManager.getServiceWSE(Constants.COUNTRY_CODE.toLowerCase(Locale.ENGLISH),
Constants.PatientService);
```

At a first glance, a possible solution to introduce a new way to access some of those properties would be to provide an implementation of the ConfigurationManagerInt interface that, just as its database counterpart does with Hibernate, would make use of some caching library/implementation (CacheImpl in the diagram) to create the caching layer. There are already some implementations available, e.g., Ehcache, Memcached.

An analysis conducted on TSL-Sync and epsos_properties DB concluded the following:

1) Properties specific to TSL workflow, not useful for caching mechanism and no longer used after TSL-based CS shutdown:
   a) ncp.countries
   b) tsl.location.<cc>
2) Properties updated by TSL workflow that are candidates to be moved to the cache:
   a) <cc>.ConsentService.WSE
   b) <cc>.OrderService.WSE
   c) <cc>.PatientIdentificationService.WSE
   d) <cc>.PatientService.WSE
   e) <cc>.DispensationService.WSE
   f) <cc>.VPNGateway.WSE

With <cc> being the two-letter country code (e.g., it, pt, gr, at, …).

If at a first glance we could tweak ConfigurationManagerService to read/write the properties listed in 2) in the cache, leaving the other ones at the database level, one must take into account that both TSL and SMP-based CS must coexist, which means that WSE properties cannot be confined to the cache, with its storage being differentiated on a per-country level (some countries will use SMP and keep their TSL files, while other will stick to TSL files in the meanwhile). Following are some Q&A thoughts on how to keep backwards compatibility with TSL-based CS during the migration:

Q1) How to know which CS (TSL or SMP) should be used to fetch configs for a specific country?
A1.1) Check SMP for that country (GetServiceGroup for that country). If it has no info, check TSL.

As per SMP business analysis document for DeleteSignedServiceMetadata (3.4.5):
If no more ServiceMetadata information is available on the related ServiceGroup, the limited information on the ServiceGroup is nevertheless kept to allow keeping track of the previously defined administrator and the service group. Should it be deleted, it is the responsibility of the "Admin SMP" user to issue the required operation (DeleteServiceGroup).

So can we rely on an HTTP 404 answer from GetServiceGroup of a specific country?

A1.2) Have a list of countries using SMP and check SMP only for those.

Can we assume ncp.countries property as reliable? It could have the list of countries using TSL, with the rest using SMP. I don't think this is reliable (it's MS responsibility to have this updated in their own NCP). So a list of SMP adopters would be needed.

Q2) How to know which information repository (DB or Cache) to consult when invoking another

NCP?
A2.1) Rely on non-existing info in one of the repositories? I don't think so.

If there's no info in Cache for a specific country, it doesn't mean it's not using SMP, just that we haven't yet fetched its info from SMP server.

If there's no info in DB for a specific country, it may be the case that TSL-Sync hasn't yet fetched the country's TSL file from CS.

A2.2) Have a list of countries using SMP and check Cache only for those.

A2.3) Another solution suggested would be to have the cache on top of the DB, as in this workflow:

1. Check cache for config;
2. If not in cache, check DB;
3. If not in DB, fetch from SMP.

How can we be sure that if a value is absent from cache and DB, it should be fetched from SMP and not from TSL?

Nevertheless, a number of other problems exist regarding ConfigurationManager and the OpenNCP architecture in general.
In the diagram it's shown a lot of components depending on ConfigurationManager, i.e., components which actually pack a ConfigurationManager within themselves. The components are then organized into 4 main WAR files (excluding OpenATNA):

1) TRC-STS
2) epsos-ws-server
3) epsos-client-connector
4) OpenNCP Portal

Since each of these WARs packs a ConfigurationManager and we're deploying them on Tomcat, it means that each one of them will load a different instance of ConfigurationManager (even though it's a Singleton, Tomcat uses individual class loaders for webapps). Furthermore, as suggested by the OpenNCP installation manual, these artifacts should be deployed in separate application servers (one for OpenNCP artifacts and another one for the Portal). This creates a scenario where not only exists multiple ConfigurationManagers in the same server but also different ones between servers. So, would the caching layer need to be horizontal to the artifacts (and even to servers)? Or can we confine it to the module(s) that reads-writes WSE properties (Protocol Terminators? To be checked...)?
Another problem noticed was that whenever one needed to update a property in the DB, it'd need to restart OpenNCP for it to discard the old values and read the new ones. Could this be somehow related to those old values being stored in Hibernate caching session?

Would this be the point where a refactoring of the properties storage is needed? Like returning to the old properties file for the read-only properties, perhaps even with a properties file per module, leaving the DB for the read-write ones? Or even some other solution...