EUROPEAN COMMISSION
DIRECTORATE-GENERAL FOR HEALTH AND FOOD SAFETY

General Affairs
Information systems

eHealth DSI – Change Proposal (CP) form

Tracking information:

Change Proposal ID:	CP-eHealthDSI-xxx (assigned by eHeath DSI)
Change Proposal Status[1]:	New
Date of last update:	11/03/2016
Person assigned:	(assigned by eHeath DSI)

Change Proposal Summary information:

Implementation of a caching mechanism	
Submitter's Name(s) and e-mail address(es):	Please provide name and email so we can contact you
Submission Date:	11/03/2016
Component(s) or Configuration item(s) to be changed[2]:	Configuration Manager
Actor(s) affected:	If known at time of submission; enter "None" for purely editorial changes.
Specifications/documentation impacted:	Not applicable?

[1] Status options: New, awaiting assessment, rejected, authorized, in build, in test, implemented, awaiting review, closed

[2] Components: Client connector, WS server, OpenNCP Portal, epSOSWeb, OpenATNA, TRC-STS, Security Manager, TSL-Sync, TSL-Editor, TSL-Util, Protocol Terminators, TSAM Sync, Stork Plugin, cdadisplaytool, xslttransformer, tsamexporter, cdautils, epsos-util, configuration manager, epsos-common-components, e-SENS eID richclient, e-SENS eID design-main

Reason/Business justification for the change:

The implementation of a caching mechanism is necessary in order to consolidate and improve the scalability of the OpenNCP architecture:

- The current implementation of the Configuration manager lacks resources: performance issues have been identified related to memory leaks, causing degradation of the overall performance of OpenNCP:
    - Most of the problems encountered are related to non-optimized configuration and will be resolved with the solution proposed (the other problems encountered like additivity of logs, wrong log-in level…will be treated independently from this solution);
    - Update of properties which require OpenNCP to restart.
- The refactoring of the Central Configuration Services (to use SMP/SML) will also benefit greatly from a caching mechanism by decreasing SMP/SML requests:
    - In the current implementation, the configuration is retrieved from the local database.
    - The target solution described in the document impact assessment of SMP/SML delivered by DIGIT[3] proposes to look in the memory cache if the configuration from the recipient is know and valid (if so to use this configuration to send the message. Otherwise the configuration would be retrieved from the SMP and stored in the memory cache).

The purpose of this document is to describe the common solution provided with the deployment of a caching mechanism dedicated to the OpenNCP configuration properties.

This solution does not imply a change in term of architecture nor in term of specifications because it improves the configuration manager currently in place with minimal impacts (improvement of quality code, of configuration…). This solution has also limited impact on time and cost of implementation.

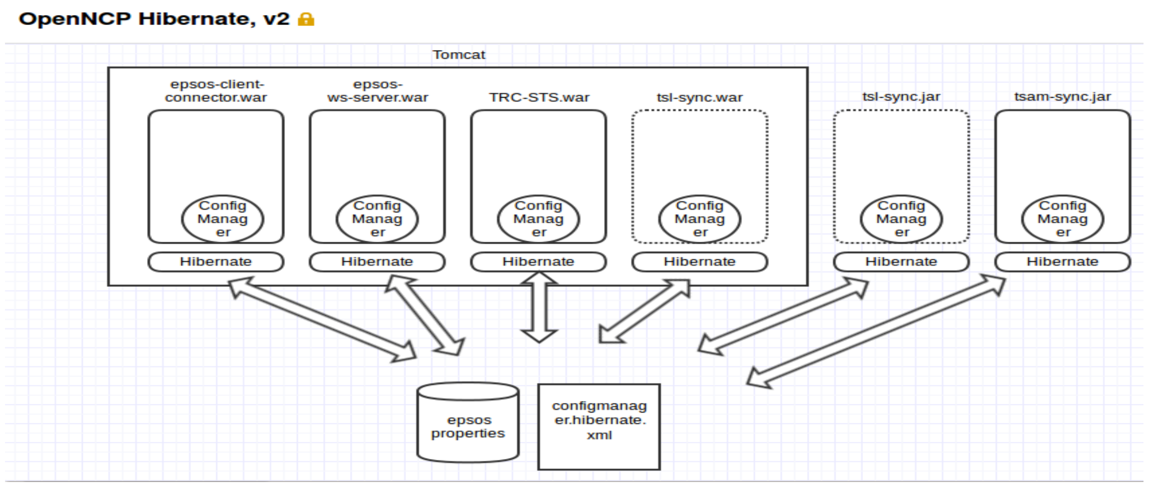
[3] "SML/SMP/eDelivery PKI Impact Assessment for the CEF eHealth DSI", p19

Description of the change (in this case "implementation")

1. Current architecture and description of the needs

The Configuration Manager is a component not designed into the original OpenNCP architecture but released during the pilot phase in order to centralize the application configuration properties into a single database (previously stored into several property files). It allows the access throughout a single Api and Object imported when it's required by the others core components.

The following picture illustrates the current architecture:



In the current architecture, the component Configuration Manager acts as a set of Singleton classes[4] that allow read/write access to the properties stored in the "openncp_properties" database. The connection configurations through which each instance of Configuration Manager connects to the database via Hibernate are stored in a main configuration file stored in the server's file system (configmanager.hibernate.xml).

Each of the following artifacts is then deployed into Tomcat and connects to the database by means of Configuration Manager:

- epsos-client-connector
- epsos-ws-server
- TRC-STS
- TSL-Sync

TSAM-Sync and, optionally, TSL-Sync as standalone applications share the same behavior. The OpenNCP Portal is not considered here as it's not part of the OpenNCP suite.

[4] a design used to permit no more than one instance of a class.

2. Possible solutions

The solution to be implemented takes account the following elements:

- Conformance with epSOS specifications;
- The need to maintain both TSL and SMP-based Central Services during some time.

The following solutions have been identified:

Solution 1: Use of embedded tools:

The main principle of the use of the embedded tools is to remove direct calls to the database by loading a large dataset in memory during application startup in order to have a very short responding time when any components access to the application properties.

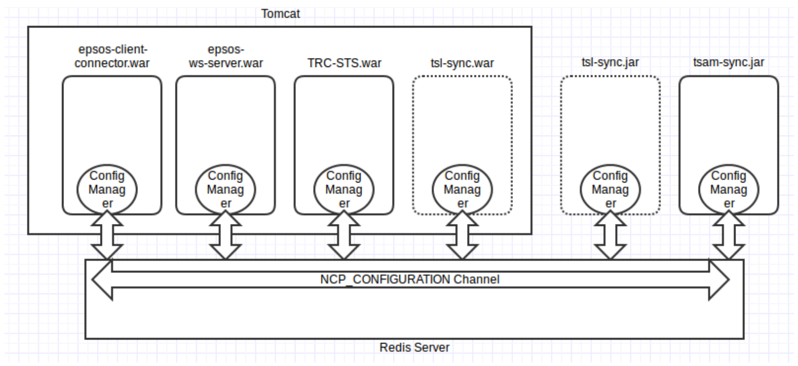
The tools considered in our analysis are Redis and MemCached.The main difference between Redis and MemCached is that Redis is a no-SQL engine (no database) while MemCached needs a database to persist the key/value pairs.

Performance tests have been conducted for both tools and a table summarizing the results of the comparison is provided in a later section.

Redis

Redis is an open source (BSD licensed) advanced key-value store, better known as a data-structure server. It supports data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps etc. It can be considered as a type of database which works with key-value pairs and uses main memory for storing data.-Its use of main memory means that it is both fast and scalable, but may be confined by the capacity of the RAM. It can be used as a No SQL database.

The following picture illustrates the possible architecture/deployment scenario:



Redis could be a good solution to implement in order to answer the needs. The central properties database could be removed.

There are still some open questions related to the security (due to the fact that it is available on the public network and that the accessibility level has to be fine-tuned to guarantee confidentiality and integrity of the data). It also complicates the installation process of an OpenNCP node. In addition the scope of this tool seems to be oversized compared to the real need of the Configuration Manager component.

MemCached:

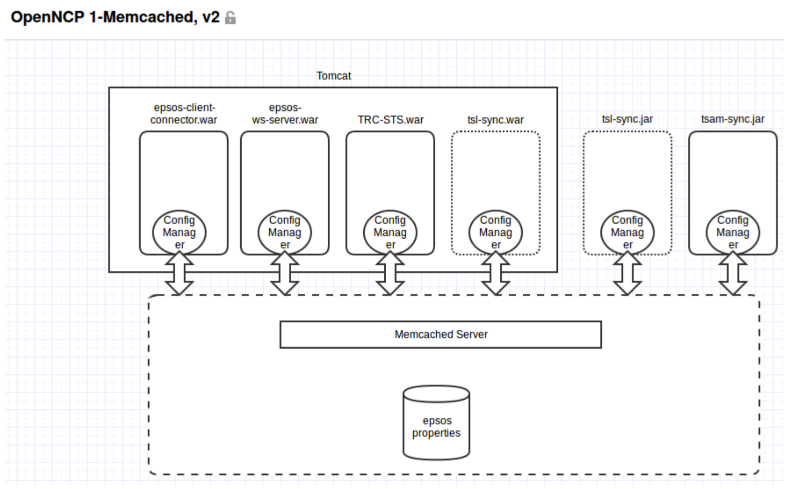
Memcached is also free and open-source software. It is licensed under the Revised BSD license.

The system caches data and objects in memory to minimize the frequency with which an external database or API must be accessed. In Memcached system, each item comprises a key, an expiration time. The cache can be seamlessly integrated with the application by ensuring that the cache is updated at the same time as the database.

Memcached performance results are the same as Redis. However documentation is lacking and there are also security issues related to the fact that it is available on a network.

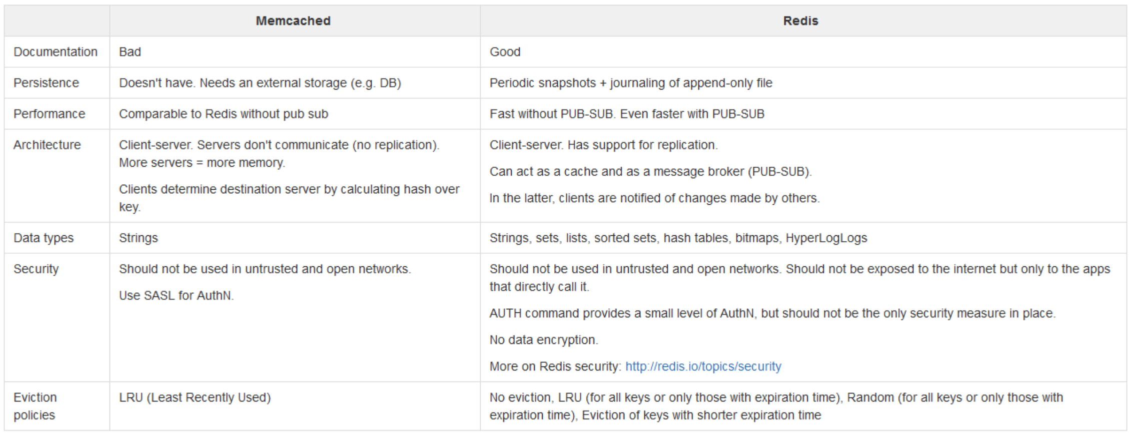
By implementing this solution the central properties database could be kept.

The following picture illustrates the possible architecture/deployment scenario:



Comparison MemCached/Redis

The performance of the two approaches has been analysed and the following table summarizes the results of the comparison:



	Memcached	Redis
Documentation	Bad	Good
Persistence	Doesn't have. Needs an external storage (e.g. DB)	Periodic snapshots + journaling of append-only file
Performance	Comparable to Redis without pub sub	Fast without PUB-SUB. Even faster with PUB-SUB
Architecture	Client-server. Servers don't communicate (no replication). More servers = more memory. Clients determine destination server by calculating hash over key.	Client-server. Has support for replication. Can act as a cache and as a message broker (PUB-SUB). In the latter, clients are notified of changes made by others.
Data types	Strings	Strings, sets, lists, sorted sets, hash tables, bitmaps, HyperLogLogs
Security	Should not be used in untrusted and open networks. Use SASL for AuthN.	Should not be used in untrusted and open networks. Should not be exposed to the internet but only to the apps that directly call it. AUTH command provides a small level of AuthN, but should not be the only security measure in place. No data encryption. More on Redis security: http://redis.io/topics/security
Eviction policies	LRU (Least Recently Used)	No eviction, LRU (for all keys or only those with expiration time), Random (for all keys or only those with expiration time), Eviction of keys with shorter expiration time

Both tools have similar results in term of performance. Both solutions imply security risks and have an impact on the architecture, adding complexity. The most relevant differienciator is the availability of the documentation in favor of Redis.

Solution 2: Refactoring the configuration manager (including the introduction of a new caching layer)

The overall performance could be improved by simply **improving the code quality of the Configuration Manager and by using some specificities of the Hibernate framework** which were not used for the time being (like the Standard Query Cache and embedded second level cache, like eHCache).

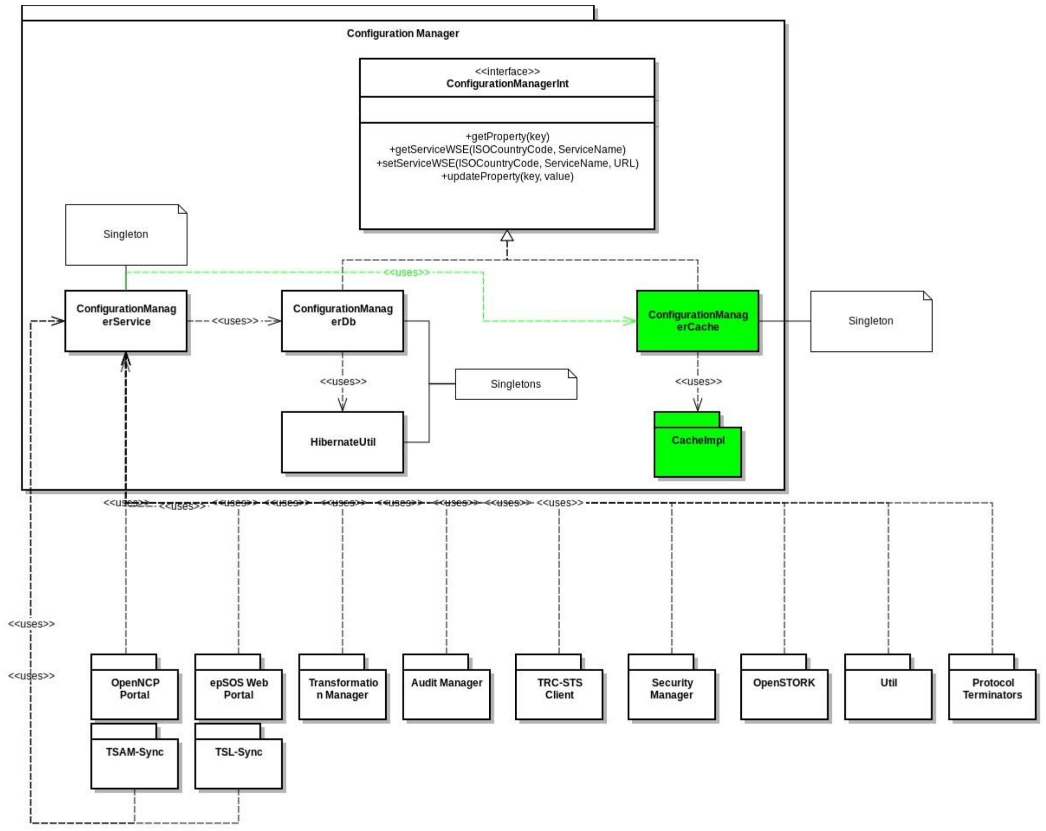
First step is to fix the configuration of the **database connection pool** with parameters adapted to the Production environment (C3p0 or DBCP libraries)
<property name="hibernate.connection.provider_class">org.hibernate.connection.C3P0ConnectionProvider</property>
<property name="hibernate.c3p0.min_size">1</property>

```
<property name="hibernate.c3p0.max_size">50</property>
<property name="hibernate.c3p0.timeout">7200</property>
<property name="hibernate.c3p0.max_statements">50</property>
```

Second step is to enable **Hibernate embedded cache** (adding the following properties into the configmanager.hibernate.xml file):

```
<property name="hibernate.cache.provider_class">org.hibernate.cache.EhCacheProvider</property>
<property name="hibernate.cache.provider_configuration_file_resource_path">/ehcache.xml</property>
<property name="hibernate.cache.use_second_level_cache">true</property>
<property name="hibernate.cache.use_query_cache">true</property>
```

In addition a **new caching layer** could be introduced, as illustrated in green in the next picture:



A new way to access and persists (Short Time To Leave) some of the properties is to provide an implementation of the ConfigurationManagerInt interface that would make use of some caching library/implementation (CacheImpl in the diagram) to create the caching layer.

Only the Java class Property have to be modified in order to enable the Cache.

The principle of this solution is to load from the database a property and store it into the Cache during a short Time To Leave (TTL) around one minute. By doing this, it will reduce significantly the number of calls to the database executed during a Health Care (HC) request. This solution is currently under testing.
If in the worst case, the result are not significant or if there are some conflicts between the different cache, it's also possible to externalize this process into a single HashMap class who will be responsible of the put/remove objects into the Cache.

This solution has a minimal impact into the source code.

3. Conclusion and implementation approach:

All solutions presented are equivalent in term of performance improvements, in relation with our needs.

In term of security, there are security risks using embedded tools like Redis or Memcache, because once they are deployed on the server, data are available throughout the network (even if those data are not healthcare data, configuration data are sensitive and must not be available to the public).
From an implementation point of view, the second solution (refactoring configuration manager, including the introduction of a new caching layer) has minimal impact on the source code and is therefore easier to implement.

As a result of the analysis, we favour the most pragmatic solution (solution 2) for the following reasons:

- The second solution implies minor changes into the architecture;
- We stay away of the security issues caused by a shared caching mechanism (Redis, Memcached etc.);
- Since this solution implies minium impact on the source code, time and cost of implementation will be reduced compared to the other solutions.

Table of references

Document name	Source or link/location
Cache implementation through ConfigurationManager refactoring	https://openncp.atlassian.net/wiki/x/EgCbB