SOCH

– the first Cultural Heritage Semantic Web

Author: Börje Lewin

Version: 1.0

Date: 2012-12-13

**Project Report from the Course
”Certifierad IT-arkitekt” no 44
at Dataföreningen Kompetens in Sweden**

Abstract

The Swedish Open Cultural Heritage (SOCH) infrastructure is the first cultural heritage semantic web. This report describes the service and identifies the most urgent non-functional requirement needed. A solution to this requirement is also presented.

Summary

One of the main objectives for the Swedish National Heritage Board (SNHB) is to make the cultural heritage available and accessible to the public. Information about the cultural heritage has always been hard to search and process. If end users are searching for information, they will realize that it is a tedious and in most cases impossible task to process this kind of information.

The purpose of this report is to describe the Swedish Open Cultural Heritage (SOCH) architecture and the architectural work of SOCH. SOCH is the first successful project to make the bulk of the Swedish cultural heritage information coherent and semantically searchable. Several crucial decisions and choices are described as well.

Some of the most important architectural principles of SOCH are as follows:

- **Semantic web and persistent URIs** – The W3 semantic web technology where Internet can be used in a more semantic way.
- **Resource Description Framework (RDF)** – RDF is the XML standard for supporting the semantic web.
- **REST-API** – The REST principles for API communication are used.
- **Harvesting technology** – Cultural Heritage data is harvested into a central repository.
- **Using a repository and implementing indexing in software** – Searching functionality is built in software.

In this report the architecture of SOCH to date is analyzed and described. The most important non-functional SOCH challenge identified in the report is to secure better availability. This is achieved by splitting the main node into two, where the secondary node is simpler than the primary node, and used for redundancy purposes only. Failover between the nodes is accomplished through Apache web-front functionality. This functionality has been developed in parallel to the report and will be implemented within soon.

Also, as a result of the analysis made when working with this report, a further recommendation for SOCH is proposed. The recommendation is to improve the tools and methods for checking XML and RDF structures, as well as the SOCH protocol adherence.

Table of Contents

Table of Figures

1 Introduction

1.1 Background

The Swedish National Heritage Board (SNHB) is the agency of the Swedish government that is responsible for heritage and historic environment issues. Their mission is to play a proactive, coordinating role in heritage promotion efforts and to ensure that the historic environment is preserved in the most effective possible manner. In the visionary documents for the SNHB, the following statements are made:

We promote knowledge and understanding of cultural heritage as an important part in people's lives and their environment, by:

- *monitoring the historic environment*
- *managing heritage statutes and surveying heritage efforts*
- *developing methods and arguments in partnership with others*
- *facilitating communication about cultural heritage*

One of the main objectives for SNHB is to make the cultural heritage available and accessible to the public. Information about the Swedish cultural heritage has always been hard to search and process. For more than 20 years different approaches have been made by different actors and tens of million SEK has been used to find solutions, but without reasonable success. Recently, however, technical development has offered new possibilities to solve this issue. The Swedish Open Cultural Heritage (SOCH) is the first service to master these challenges successfully.

Some of the most important architectural principles of SOCH are as follows:

- **Semantic web and persistent URIs** – SOCH supports the W3 semantic web technology where Internet can be used in a more semantic way, i.e. objects and relations can be narrowed down to semantic meaning. Every object has its own persistent URI, its own address.
- **Resource Description Framework (RDF)** – RDF is the XML standard for supporting the semantic web.
- **REST-API** – The SOCH API is a read-only API and can therefore use the light REST principles for API communication.
- **Harvesting technology** – Cultural Heritage data is harvested into a central repository as opposed to using real-time federated searching principles.
- **Using a repository and implementing indexing in software** – A simple central repository holds the cultural heritage objects and the searching functionality is built in software, not in a database.

1.2 Problem Description

The problem is best described through an example. If an end user[1] is searching for information about stone axes from the 13th century, found in Stenkyrka on Gotland, he or she is (or was before the evolution of SOCH) forced to work in the following manner:

- Use search engines (e.g. Google) to get a lot of hits. However, only a very small fraction would be useable, since a search like that is non-semantic by its very nature, especially since the information itself is also non-semantic, or

- Conduct searches at different cultural institutions like:

    - Museum of Gotland
    - National Historical Museum
    - LIBRIS
    - Fornsök
    - Local heritage institutions and community centers
    - Etc.

It is a tedious and in most cases impossible task to process this kind of information.

1.3 Objective

The objective of this report is to describe the SOCH architecture and the architectural work of SOCH. SOCH is the first successful project to make the bulk of the Swedish cultural heritage information coherent and semantically searchable. Several crucial decisions and choices are described as well.

The most important operational and/or developmental challenge is identified, and a solution to handle this challenge is proposed.

1.4 Scope

The different cultural heritage institutions delivering data to SOCH (designated “Content Providers”, below) are not described in this report. Nor are the applications on national level or within Europe (www.europeana.eu) using the SOCH API (described in chapter 4) or the SOCH harvesting mechanisms. Some references to Europeana are included in the report though. Please see the glossary for more information about Europeana.

[1] a person working with research relating to the cultural heritage or a private person who is interested in the field, using Internet to find information

1.5 Audience

The intended audience of this report is:

- Students at the course ”Certifierad IT-arkitekt”
- People from different parts of Europe who work to make the cultural heritage available for the public, which includes people in both technical and business roles
- People interested in the semantic web and the Resource Description Framework (RDF)

The reader is preferably familiar with the semantic web (sometimes referred to as “Web 3.0”) and RDF. These terms are explained in the attached glossary, but a deeper knowledge will provide the reader with more understanding of this report. A short description of semantic web reasoning is included in chapter 3.

1.6 Method

This report analyzes and describes the architecture of SOCH to date and also proposes further development of this software infrastructure. It presents interviews, analyzes code and documentation, as well as making use of concrete knowledge from the course literature and utilizing this knowledge for the problem at hand.

2 Previous Efforts in this Area

As mentioned above, for more than 20 years efforts have been made to build solutions for extensive search of cultural heritage information. Some of the reasons that these attempts have failed are the following:

- mature and developed technology not yet available
- questionable architectural choices, for example:
  - Extensible Messaging and Presence Protocol (XMPP) strategy instead of the ubiquitous HTTP
  - federated searching strategy
- not enough momentum and drive
- not enough committed stakeholders included in the efforts
- development resources provided, but no provision for continued operation and maintenance
- politics

3 Semantic Web

Since one of the most important architectural principles of SOCH is the adherence to the semantic web, a short introduction to the subject is presented here.

The semantic web is sometimes also called *Web 3.0* or *Web of Data*. The word “semantic” implies that we add meaning to the web. It is not just a chunk of data, but structured or semi-structured information.

Let’s make an analogy. Think about the difference between searching in a file system and searching in a database. When you search in the database you can get a semantically correct set of data, e.g. all system developers between 50 and 55 years old. When you search in a file system (e.g. “system developers 50 51 52 53 54 55”) you can get just about anything containing one or more of the strings in your search phrase.

Now apply this thought to the web of today. When searching through the main search engines you can get a lot of hits. Search for stone axes from the 13th century, found in Stenkyrka on Gotland in one of these search engines and try to use the result for further processing. It is just not possible.

The W3C[2] Semantic Web is a way to use the web as a database. A strict XML format encapsulated by RDF and persistent identifiers for objects (so called “persistent URIs”) are some of the main technologies used.

It is not possible to describe the semantic web thoroughly in this report (please see the glossary and other sources), however, here is an example based on the cultural heritage context:

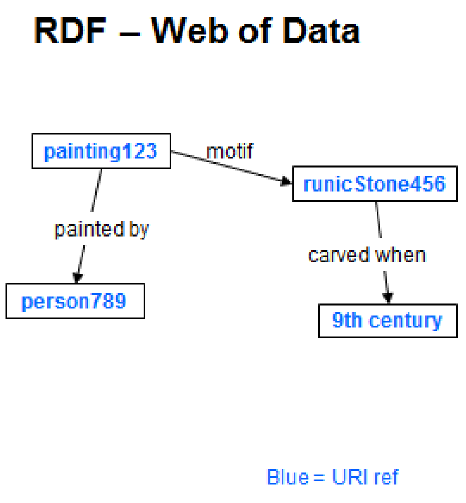

Figure 1: Web of Data (1)

Let’s start out with a painting made by a painter with a motif of a runic stone carved in the 9th century. Note that the objects are URI references, i.e. they have persistent URI identifiers on the Internet.

---

2 http://www.w3.org/2001/sw/

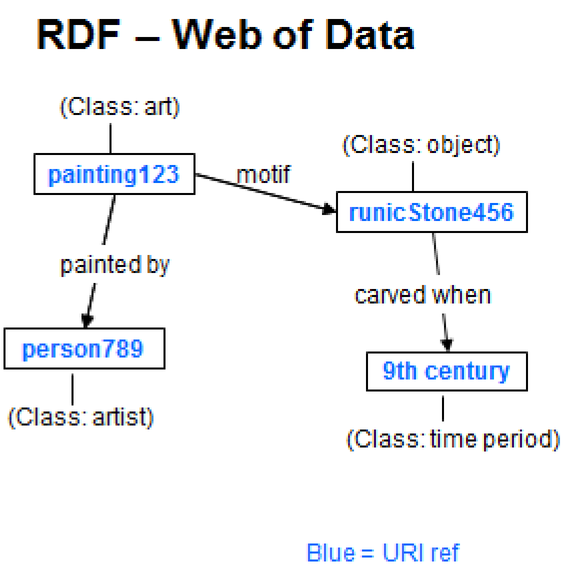

Figure 2: Web of Data (2)

Using RDF we can couple instances to classes just as we do in object oriented design.

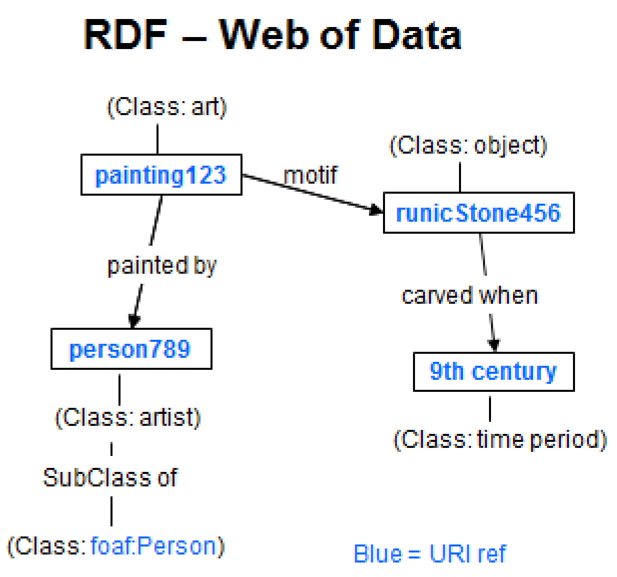

Figure 3: Web of Data (3)

Here we state that artist is a subclass of foaf:Person, which is a de facto standard for personal data on the semantic web. **Now other semantic webs potentially connecting to our own, will understand that person789 is in fact a person.**

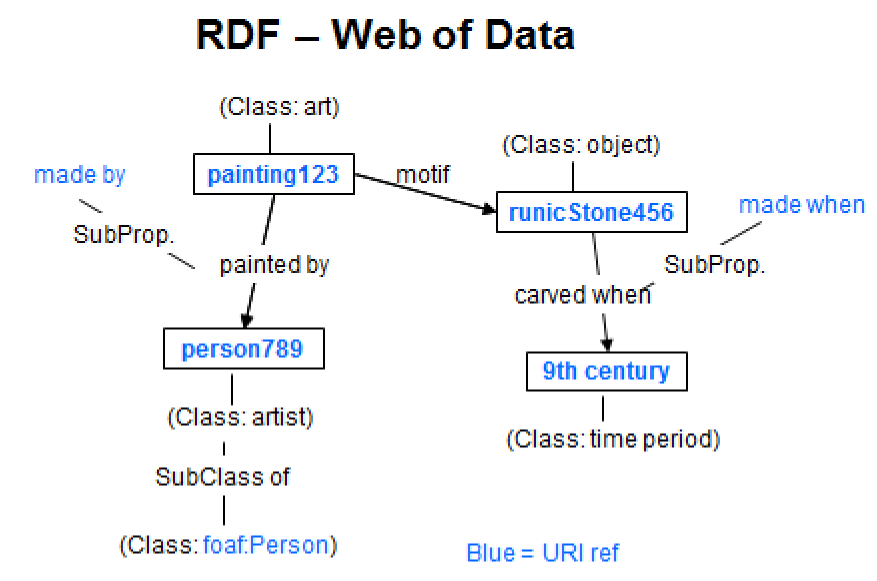

Figure 4: Web of Data (4)

Not only objects, but also the relations can be locked into an agreed semantic meaning.

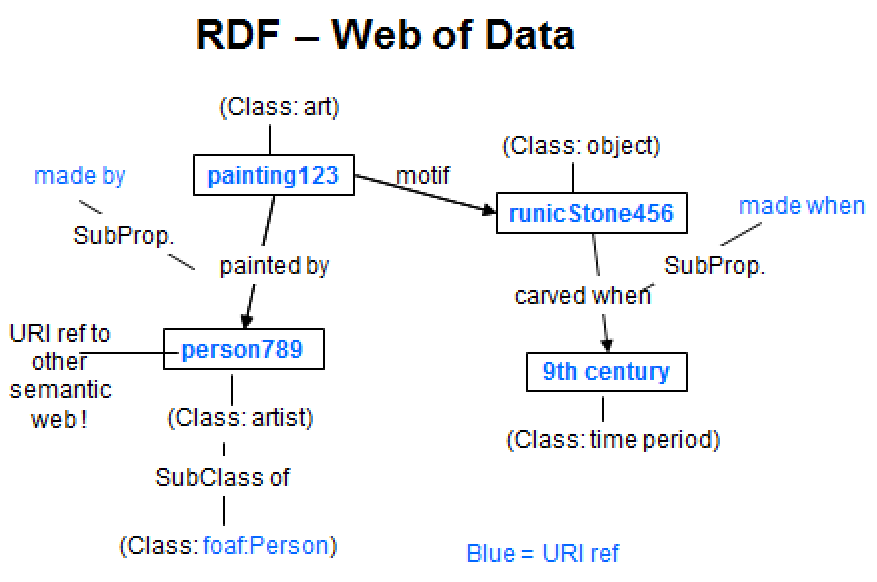

Figure 5: Web of Data (5)

Objects can also be directly pointing (using the **sameAs** property which is well known and of great importance in semantic web technology) to other objects in any semantic web, representing that very same object.

The RDF for this example is as follows:

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns="http://kulturarvsdata.se/ksamsok#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:foaf="http://xmlns.com/foaf/0.1/">

   <art rdf:about="http:/kulturarvsdata.se/example/painting123">
      <motif rdf:nodeID="runicStone456"/>
      <painted_by rdf:nodeID="person789"/>
      <label>Runic stone in beautiful light</label>
   </art>

   <object rdf:nodeID="runicStone456">
      <carved_when rdf:nodeID="period"/>
      <label>Nice Stone</label>
   </object>

   <artist rdf:nodeID="person789">
      <foaf:name>Martin Douglas Jr</foaf:name>
   </artist>

   <timeperiod rdf:nodeID="period">
      <time>9th Century</time>
   </timeperiod>
</rdf:RDF>
```

Now validate this xml structure online at http://www.w3.org/RDF/Validator/ and you will see the RDF triplets that could be stored in a triple store (three column database) to represent the structure. You can also get a graph representation from the validator:

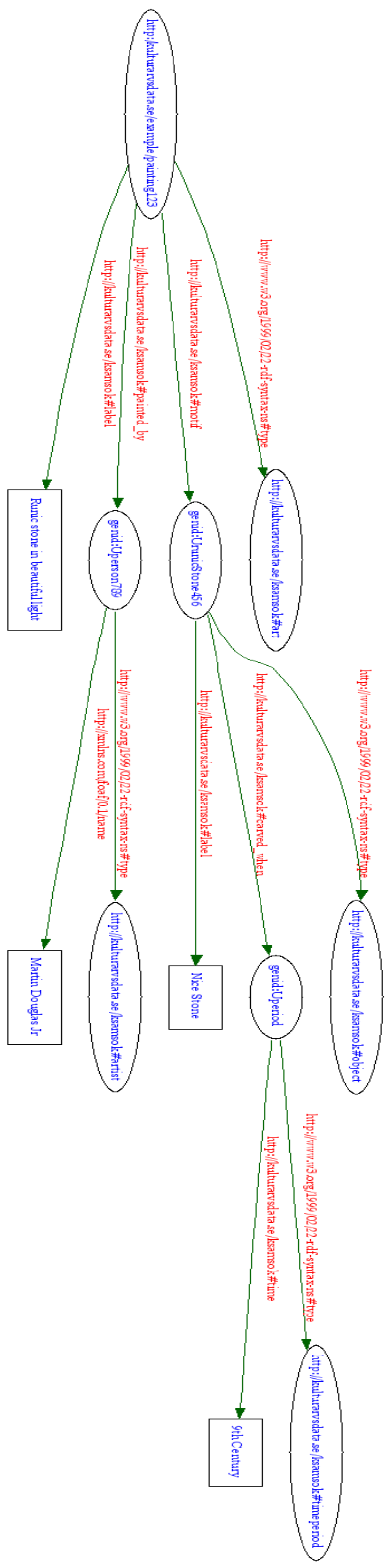
http://kulturarvsdata.se/example/painting123
http://kulturarvsdata.se/ksamsok#label
http://kulturarvsdata.se/ksamsok#painted_by
http://kulturarvsdata.se/ksamsok#motif
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
Runic stone in beautiful light
genid:Uperson709
genid:UrunicStone456
http://kulturarvsdata.se/ksamsok#art
http://xmlns.com/foaf/0.1/name
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
http://kulturarvsdata.se/ksamsok#label
http://kulturarvsdata.se/ksamsok#carved_when
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
Martin Douglas Jr
http://kulturarvsdata.se/ksamsok#artist
Nice Stone
genid:Uperiod
http://kulturarvsdata.se/ksamsok#object
http://kulturarvsdata.se/ksamsok#time
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
9th Century
http://kulturarvsdata.se/ksamsok#timeperiod

Figure 6: W3 RDF Graph Generation

Note that the graph is slightly modified in order to be somewhat more readable. If you can't read it, please just validate the XML yourself! The graph is also attached as Appendix A.

4 Basic Architecture

The basic architecture of SOCH is shown below.

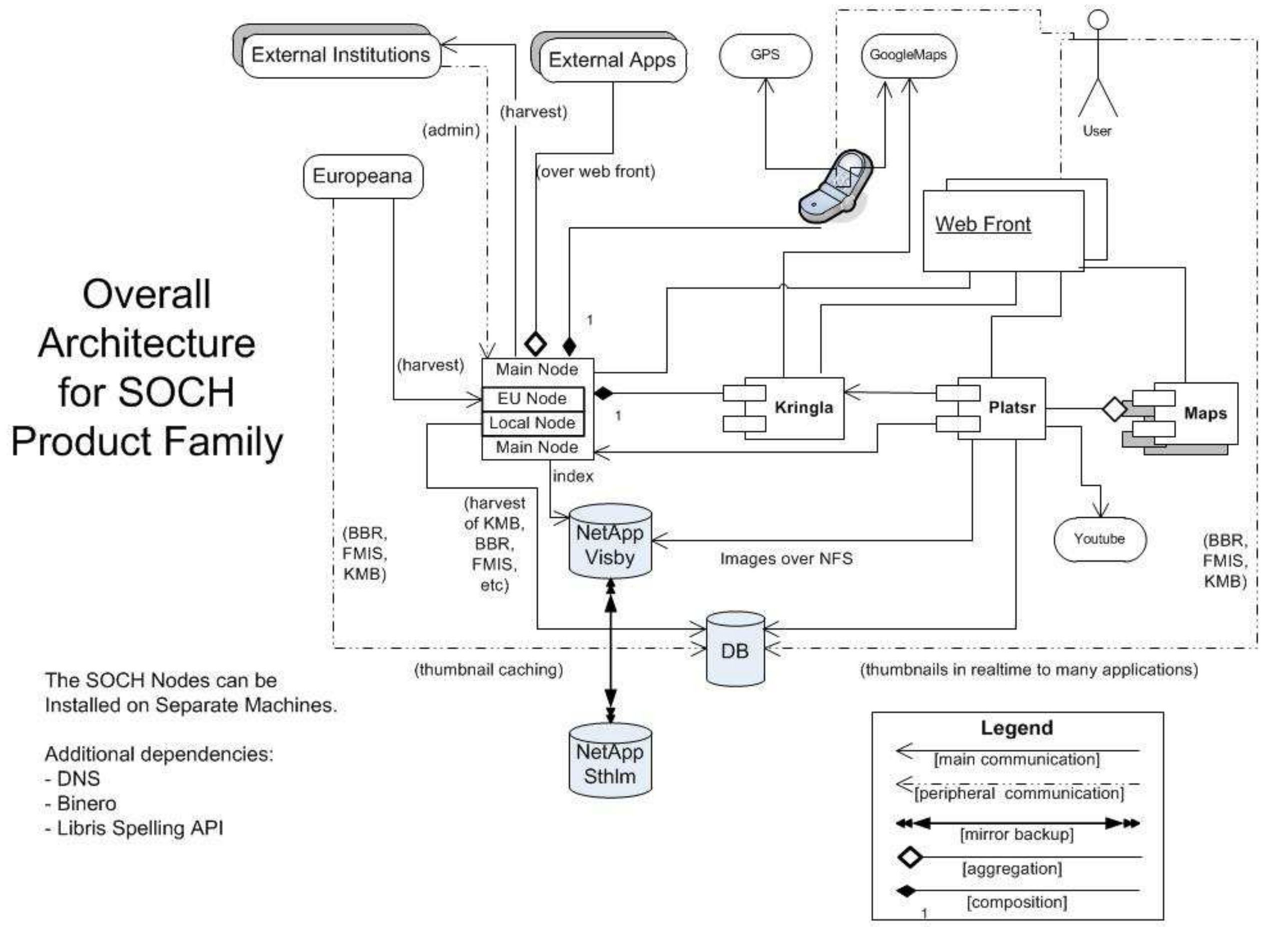

Figure 7: Overall Architecture (Communication View)

Some further explanation of figure 7 follows:

- **Platsr** is a SNHB web application for documenting the cultural heritage of today. Users can upload pictures and write stories about different places, mainly within Sweden but also abroad. The web site is found at www.platsr.se and has a Facebook inspired user interface. Places registered in Platsr can be linked to SOCH objects, e.g. an ancient monument close to the place that a user describes in text and photographs.

- **Kringla** (www.kringla.nu) is the main window towards SOCH; please see the Glossary for more information.

- **Maps** is a term that refers to map servers provided by SNHB. These servers provide background maps and different layered maps of Sweden.

- **NetApp** is a commercial data storage system with a mirroring backup solution. In this figure it denotes that the SOCH index is backed up in another location. However, if the index is lost, it is easy to rebuild it. This would take a day or two.

- **DB** refers to the SNHB in-house RDBMS databases hosted by Oracle and PostgreSQL.

- **FMIS, BBR, KMB** are SNHB databases that deliver data to SOCH just as external institutions do. Please see the Glossary for more information about these databases/systems.

This overall architecture view is a bit crowded but has served well internally in the organization. It is mainly a communication view, but it gives hints on logic and deployment as well. The main characteristics of the system is to harvest data from a long range of cultural heritage institutions, store the data as semantic objects in a simple repository, and make the objects available for users and applications (through the SOCH API) using an open source indexing machine.

Maybe the most important aspect of the SOCH infrastructure is the SOCH API. In figure 7 the API is used by external applications, Kringla, and mobile applications. It is a simple REST API for searching only, and it is fully documented at www.ksamsok.se.

With the SOCH API, a search for urns found in Skåne would be specified as follows:

http://kulturarvsdata.se/ksamsok/api?stylesheet=..%2Fstylesheet%2FsearchStyle.xsl&x-api=test&method=search&hitsPerPage=250&query=item= "urna"+and+place="skåne"

This will return objects from four different institutions. Leaving out the stylesheet parameter in the API request will return the RDF structure as follows:

```
<result>
  <version>1.0</version>
  <totalHits>331</totalHits>
  <records>
    <record>
      <rdf:RDF>
        <Entity rdf:about="http://kulturarvsdata.se/shm/object/96927">
          <ksamsokVersion>0.99</ksamsokVersion>
          <buildDate>2010-11-26</buildDate>
          <collection rdf:resource="http://mis.historiska.se/rdf/collection#0"/>
          <createdDate>2002-12-01</createdDate>
          <lastChangedDate>2002-12-01</lastChangedDate>
          <serviceOrganization>shm</serviceOrganization>
          <serviceName>object</serviceName>
          <url>http://mis.historiska.se/mis/sok/fid.asp?fid=96927&g=1</url>
          <subject rdf:resource="http://kulturarvsdata.se/resurser/Subject#archaeology" />
          <mediaType>text/plain</mediaType>
          <dataQuality rdf:resource="http://kulturarvsdata.se/resurser/DataQuality#raw" />
          <itemType rdf:resource="http://kulturarvsdata.se/resurser/EntityType#object" />
          <itemLicense rdf:resource="http://kulturarvsdata.se/resurser/License#pdmark" />
          <itemName rdf:nodeID="name001" />
          <itemLabel>urna av lera</itemLabel>
          <itemSpecification rdf:nodeID="spec001" />
          <itemDescription rdf:nodeID="desc001" />
          <itemMaterial rdf:nodeID="mat001" />
          <itemNumber rdf:nodeID="num001" />
          <itemMeasurement rdf:nodeID="meas001" />
          <context rdf:nodeID="ctx001" />
          <context rdf:nodeID="ctx002" />
          <isPartOf>http://kulturarvsdata.se/shm/inventory/20257</isPartOf>
          etc....
```

4.1 Database (Repository)

The database is called a repository since it only holds information about cultural objects that is harvested from the different content providers. The original data is still stored by the content providers. The repository is basically just one simple database table and the structure is as follows, which is actually a nosql type of solution:

- URI is the persistent URI for the object.
- OAIURI is the URI used for harvesting purposes (SOCH is aggregated from many content providers).
- SERVICEID is an identifier for the institution/database part of the URI.
- XMLDATA is the whole XML chunk for each object. These are the actual “documents” in the context of document/nosql database technology.
- CHANGED keeps track of when the specific object was most recently changed.
- ADDED specifies when the object was added to SOCH (newly registered object OR material from a new content provider connected to SOCH).
- DELETED: SOCH supports harvest updates where you need only to harvest deleted, changed or added objects. Europeana does not support this, so there is a need to keep track of what objects have been deleted in order to provide Europeana with that information.
- DATESTAMP specifies when the object was most recently harvested. This information is needed in order to update only objects that have been changed after a specified date.
- STATUS is set to 1 while a record is due for re-harvesting. If the status is still 1 after re-harvesting, this implies that the record (object) has been deleted in the original database. This triggers setting the DELETED column to true.
- IDNUM is just an internal ID.

4.2 Scenarios (User Perspective)

The user perspective of SOCH is best shown in the following use case diagrams:

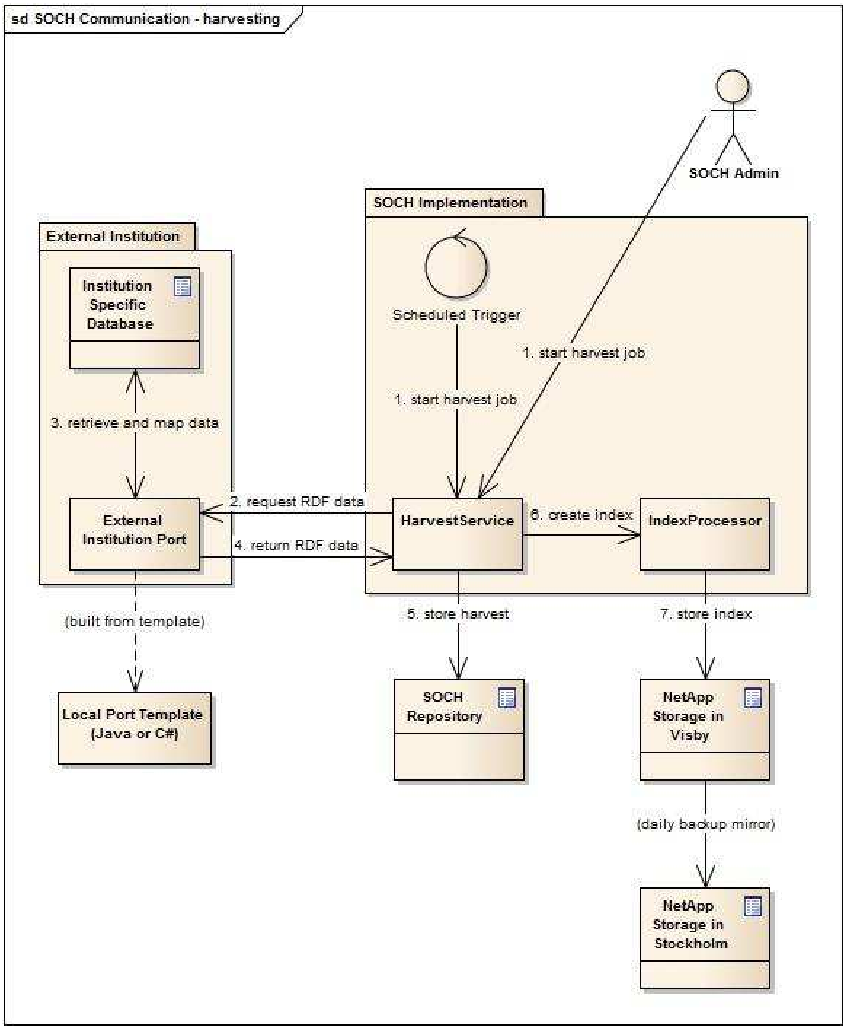

Figure 8: Harvesting performed by user or system

The SOCH admin can initiate harvesting of any service, but the normal procedure is to use scheduled harvesting. Some content providers are harvested every night, others every week or month. Note that only metadata is harvested. Thumbnails and other digital resources, as well as the objects' main presentations, are located at the content providers' web sites.

Some further explanation of Figure 8:

- The SOCH implementation is split in two components, one for harvesting purposes and one for indexing. The latter includes the Java Solr[3] open source package.

- **SOCH Repository** is the PostgreSQL database for storing the cultural heritage object metadata.

- **External Institution Port** is the software used by institutions to communicate with SOCH. The local port should be constantly listening for server harvesting requests.

[3] See Glossary for more information

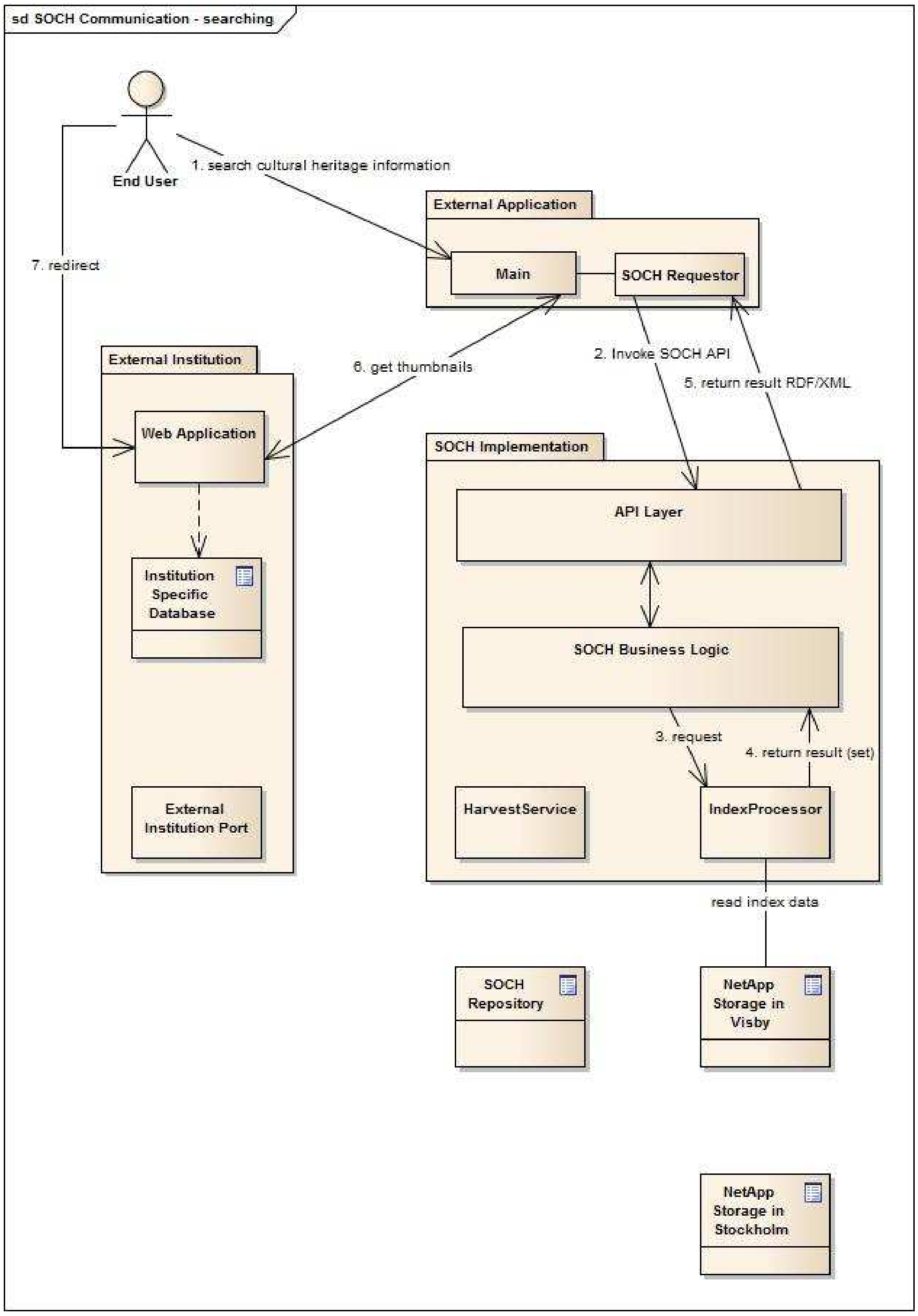

Figure 9: Searching performed through an application (e.g. Kringla) and the SOCH API

Applications use the SOCH API to search and carry out a number of operations. This REST API is described at http://www.ksamsok.se/in-english/api/

Note that the SOCH Repository is not needed by external applications or any SOCH API usage since the full RDF content is also included in the index.

4.3 Deployment View

The deployment is explained in the following view:

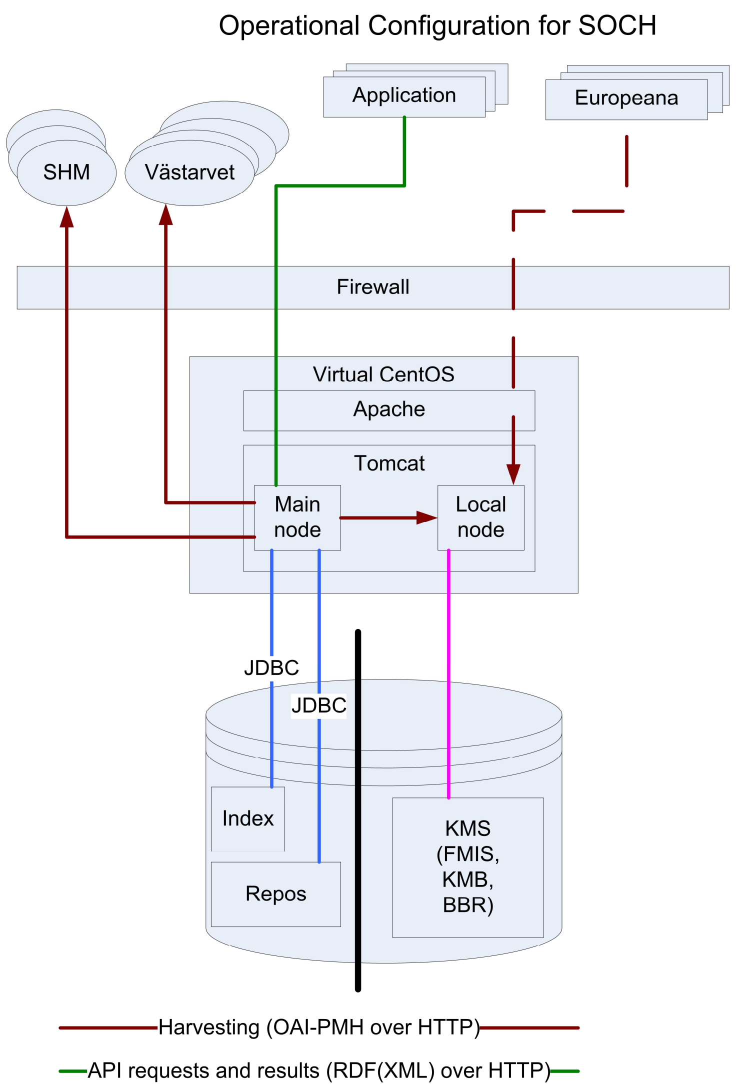

Figure 10: Deployment and Operational Configuration

SOCH is implemented in Java and included in a Tomcat container on a CentOS (Linux) virtual machine. Some further explanation of figure 10 follows:

- **Index** is the Java Solr index used for searching SOCH objects.
- **Repos** is the repository where all the harvested objects are stored. It is used when the index is created or updated, but it is not needed for the system to operate as a web service or API interface since the full data content of the objects is actually also included within the index.
- **SHM** is the National Historical Museum of Sweden. The museum is only one example of cultural institutions harvested by SOCH.
- **Västarvet** is an organized set of cultural institutions, and only one example of cultural institution groupings harvested by SOCH.
- **OAI-PMH** is the transport protocol; please see the Glossary for more information.

4.4 Harvesting States

When harvesting data from different institutions, their datasets (services) go through a limited number of stages:

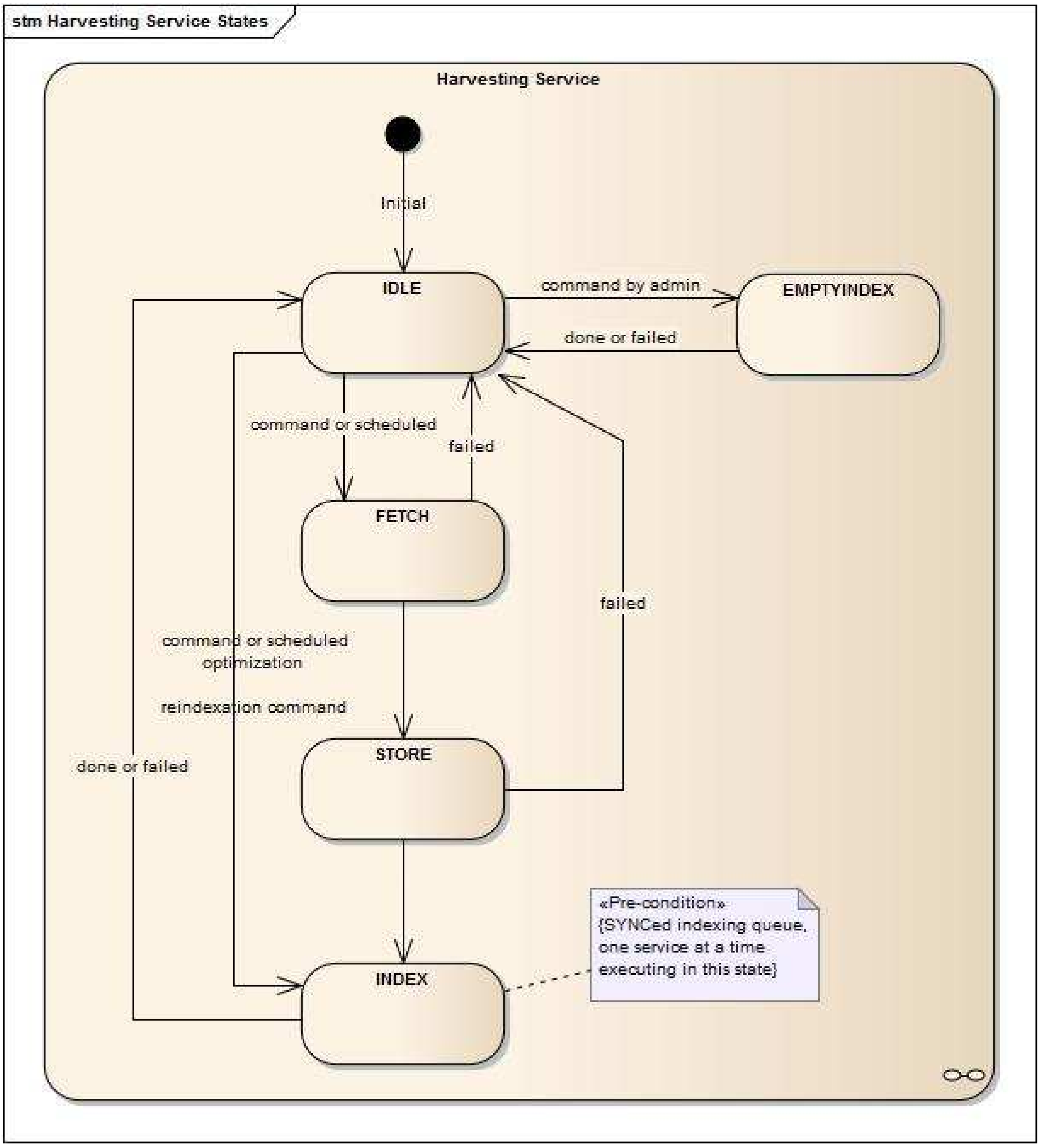

Figure 11: Harvesting States

Figure 11 shows the harvesting states that a service goes through when harvested to SOCH. A service is a well defined set of objects from one cultural institution (e.g. photos from the museum of technology) where all these objects are uniquely identified. Normally they are harvested from one and the same database table. The object information is first fetched by administrator command or by scheduled harvesting. When the whole set of object are retrieved without errors, they are stored in the database repository. If also the storage procedure is successful, the set of objects are indexed and immediately searchable.

5 Important Architectural Choices and Decisions

In this chapter some of the most important ongoing or previous architectural choices and decisions are described.

5.1 HTTP or XMPP?

XMPP is an open-standard communications protocol for message-oriented middleware based on XML. Previous attempts by external forces to meet the challenges at hand, tried to use XMPP and a sophisticated system of autonomous nodes. This strategy was abandoned when designing the architecture of SOCH, mainly for the following reasons:

- Autonomous nodes are never truly autonomous. Operational and support resources are needed and must be planned for and financed. The cultural heritage institutions are very skilled at handling cultural heritage tasks, but do not have (nor should they have) advanced IT system development skills.

- For SOCH to be successful it must attain a critical mass of interested organizations and other stakeholders. For this to happen it must be very easy to deliver data, a simple HTTP solution is required.

5.2 Federated Search or Harvesting?

SNHB has built a reference implementation that shows how the SOCH infrastructure can be used. This implementation is named Kringla and can be seen as one of many possible presentation tiers of the SOCH system. A previous effort of SNHB was named “Kulturmiljösök”, and in a way it was a predecessor of Kringla. The end-user would use a web form to enter search criteria, after which a federated search was made to a handful of different content providers. These providers had different APIs or other means of communication. The results from some of the sources were returned in a few seconds and the slowest providers were responding after several minutes. The user had to choose whether to wait for the final results or settle for the results returned “so far”.

The first approach suggested for SOCH was to use federated search but with a common, agreed protocol. The thinking was that the continuous evolution of technical infrastructure would make it possible to return results more quickly in a new system.

However, such a system would be very dependent on all (or most) of the providers always being available and accessible, and the cultural heritage sector does not have the resources to act as a 24/7 information providers. How, then, can the system be built with a minimum dependency on the different providers' own systems?

The “Use an intermediary” tactic [Bass] was the architectural solution used instead of federated search. The mechanism is called “Harvesting” and means that all the content providers’ data is fetched/harvested, possibly at night time, to a central repository. The investment for different providers in order to join the SOCH community is limited to building a local adapter (port) that translates the local database format into the protocol agreed upon by SOCH and sends the data to the repository on request (pull principle).

This architectural choice can also be backed up by the following patterns [Buschmann] on the client side:

- Interpreter – The local port at the content provider interprets the local database structure and semantic into the SOCH protocol. (Note: not interpretation of commands, but of data and data structures)

- Template Method – Not only a method but rather the whole local port implementation can be built from templates used by previous implementations. This is a component re-use and adaptation.

- Declarative Component Configuration – The SQL statements needed to fill SOCH elements can be declared in a separate configuration file, keeping the programming code free of the local adaptations. This is a separation of concerns that supports modifiability.

5.3 Object Relationships in Both Directions?

The semantic web principle relies heavily on relations between semantic objects being perfectly and solidly defined so that different systems will interpret the relations in the same way. If we in the SOCH community define the relation "soch:surname", this relation should be narrowed down semantically by defining it as a sub-property of, or sameAs relation to, some globally accepted predicate in some globally accepted namespace, e.g. "foaf:lastName[4]". In this way all the different semantic (RDF) webs could interoperate. This is somewhat implied in expressions like "Web of Data" and "Linking Open Data (LOD)".

[4] In this example, if surname and lastName are semantically equivalent, foaf:lastName should be used in the SOCH protocol and soch:surname should be discarded.

In SOCH 1.1, the possible object relations are shown in the figure below.

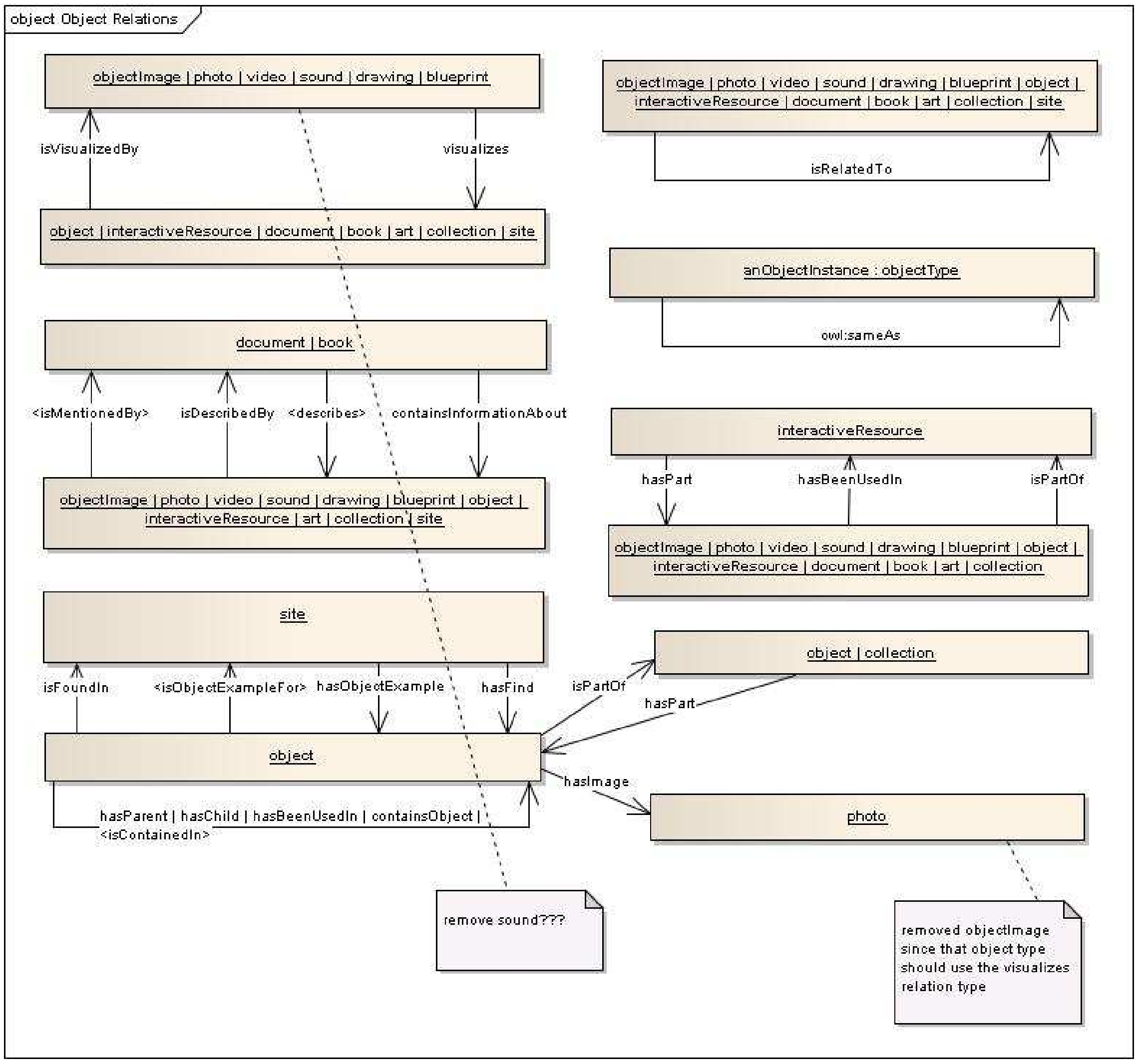
object Object Relations
objectImage | photo | video | sound | drawing | blueprint
isVisualizedBy
visualizes
object | interactiveResource | document | book | art | collection | site
objectImage | photo | video | sound | drawing | blueprint | object | interactiveResource | document | book | art | collection | site
isRelatedTo
anObjectInstance : objectType
owl:sameAs
document | book
<isMentionedBy>
isDescribedBy
<describes>
containsInformationAbout
objectImage | photo | video | sound | drawing | blueprint | object | interactiveResource | art | collection | site
interactiveResource
hasPart
hasBeenUsedIn
isPartOf
objectImage | photo | video | sound | drawing | blueprint | object | interactiveResource | document | book | art | collection
site
isFoundIn
<isObjectExampleFor>
hasObjectExample
hasFind
object | collection
isPartOf
hasPart
object
hasImage
photo
hasParent | hasChild | hasBeenUsedIn | containsObject | <isContainedIn>
remove sound???
removed objectImage since that object type should use the visualizes relation type

Figure 12: Object relationships in SOCH

The problem with relations is how to make them bidirectional. If a content provider creates a relationship from one of its objects to an object owned by another content provider (e.g. a museum connecting an object to a book owned by a library institution which is also a SOCH content provider), this is no problem. They just use the persistent URI for the literature object and specify the relation as “isDescribedBy”. However, users or applications browsing information about the book will not see this relationship; they will not see that a museum object is connected.

In a fully developed web of data, RDF and SPARQL (the query language for RDF)[5] will handle all this, so that relations can be found in both directions even if they are only declared for one of the directions. This is part of the semantic web architecture. However, so far very few web applications make full use of this technology.

[5] SPARQL is outside the scope of this report, see the glossary for more information

Some possible solutions to this challenge are:

- Use automatic or manual update of the library institution's local database. Automatic update from external sources would not be accepted by many institutions, which obviously want to be in control of their data. Manual update is not feasible; the institutions would create work for each other in a way that would never be accepted, and even if it were possible, it still would mean a lot of latency; the back direction of the relation would be implemented maybe half a year after the original link was made. There is also a similar problem as soon as a relation is changed or removed. This approach is considered to be impossible.

- Store the bidirectional information in a central database. The problem with this approach is that it introduces primary data to the central node. A repository with data copied from the content providers is secondary data that doesn't even need to be backed-up, but a central database should be avoided if possible, since it would introduce new dependencies and new components in the infrastructure.

The architectural solution found and chosen is to make use of the Solr[6] index and a new API method. This is a simple solution that is virtually free of charge since SOCH already uses a Solr index to find objects with certain values for certain parameters. Just search for objects that have a relation to the object at hand, return these objects to the application, and then leave it to the applications to search for the relations. In the resulting data set, mark the relations found in other objects with "deduced" in case the application wants to separate relations belonging to the object at hand from relations found in other objects.

For a description of the API method and a URL to see the result for a live object, see http://www.ksamsok.se/api/metoder/#getRelations.

A result example from using the getRelations API method:

```
<result>
  <version>1.0</version>
  <relations count="204">
    <relation type="isVisualizedBy">http://kulturarvsdata.se/raa/kmb/16000300035356</relation>
    <relation type="isVisualizedBy">http://kulturarvsdata.se/raa/kmb/16001000013431</relation>
    <relation type="hasPart" source="deduced">http://kulturarvsdata.se/shm/object/882213</relation>
    <relation type="hasPart" source="deduced">http://kulturarvsdata.se/shm/object/882211</relation>
    <relation type="hasPart" source="deduced">http://kulturarvsdata.se/shm/object/882210</relation>
    <relation type="isVisualizedBy">http://kulturarvsdata.se/raa/kmb/16001000013436</relation>
    <relation type="isVisualizedBy">http://kulturarvsdata.se/raa/kmb/16001000013438</relation>
….
```

5.4 Splitting Components

Originally, SOCH was one central component. In later designs, the indexing part has been lifted out to a separate component, see e.g. figure 8 where the harvestService and the indexProcessor are two separate components. This adheres to the Domain Model pattern [Buschmann].

5.5 What can we do about the SOCH Dependency on the Database?

By adding the whole RDF structure for the objects in the Solr index, the repository can be made superfluous in normal operation. This means that the SOCH infrastructure runs without using the repository. However, the repository is still needed for building and optimizing the index.

[6] Solr is an open source enterprise search platform from the Apache Lucene project (see also Glossary)

5.6 *Implementing Persistent URIs for objects*

Implementing persistent URIs for SOCH objects was really just a three-step procedure:

a) Decide what the URIs should look like, and how to build them uniquely

b) Store these URIs along with the objects in the repository

c) Make sure that the web server returns the object data as an XML structure for a given URI

Uniqueness

The simplest way to set unique URIs is to re-use the objects' already unique identifier in the source database. If the database is old and the object IDs change automatically, e.g. at object removal, then this is not sufficient. The content provider would have to insert an extra column for object unique identifiers in this case.

An example principle for a persistent URI is the following:

http://selfcontrolled_domain/institution/database/object_ID

Here is an existing URI that adheres to the above principle:

http://kulturarvsdata.se/shm/object/rdf/119357

Persistence

Persistence means that the URI is valid and unique over time. If you click on the URI above, you can see that the XML data includes the following elements:

<thumbnail>
<url>
<lowresSource>

These are the actual links to the object on the web and to different image resolutions. These filenames, domain names, and parameters can change over time. In fact, a few years from now the URLs could have been changed completely. However, as long as this is reflected in the XML data, people and machines will always find the resources anyway, simply by trusting the URI as the first contact, the bootstrap.

Providing answers for the URIs on the Internet

How do you accomplish an answering machine that will return the XML data for any valid URI?

Firstly, you create a column in the OAI-PMH repository to keep the URI for each object.

Secondly, you see to it that all requests to the web server that are URIs will be distinguished from all other traffic. For SOCH, the web front follows these rules:

- If the domain name is followed by /oaicat-europeana, the request is a Europeana harvesting request and will be handled as such.
- If it is followed by /ksamsok, the request is an API request for the SOCH API.
- If it is followed by /resurser, it is a request for SOCH controlled vocabularies or authorities.
- In all other cases, the database is searched for a URI matching the incoming request. If found, the XML data is fetched from another column in the same database and returned over HTTP.

5.7 *Design of a new Module for User Generated Content (UGC)*

One of the latest additions to the SOCH infrastructure is the capability to add user generated content, entering the field of crowd sourcing. The idea is that not only institutional date should be visible and maintained, but also additions from users of the web.

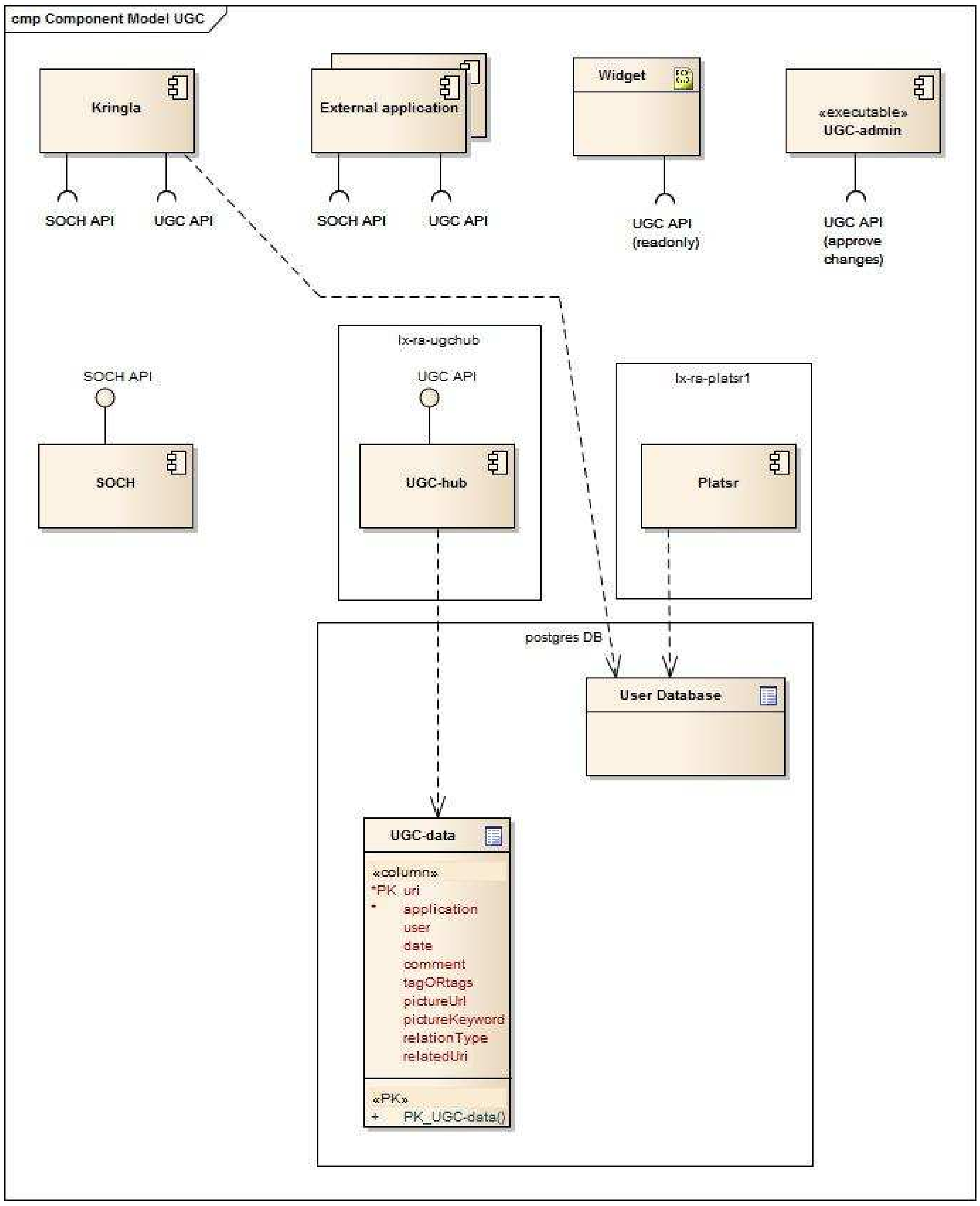

Figure 13: Addition of UGC Hub with Loose coupling

The UGC hub has a minimum of dependencies. It can only be reached through an API and only stores data in a simple database table.

6 What is the Most Important Challenge for SOCH?

6.1 Identification of Challenge

Let’s investigate the different quality attributes according to [Buschmann] (operational and development properties) and [Bass] (system qualities):

- ➢ Operational properties according to [Buschmann]:

  Stability

  Scalability

  Performance

  Availability

  Security

- ➢ Development properties according to [Buschmann]:

  Maintainability

  Extensibility

  Adaptability

  Reusability

- ➢ System qualities according to [Bass]:

  Availability

  Modifiability

  Performance

  Security

  Testability

  Usability

- ➢ Let’s also add the following:

  Interoperability

  Portability

This all results in the following gross list:

- Adaptability
- Availability
- Extensibility

- Interoperability
- Maintainability
- Modifiability
- Performance
- Portability
- Reusability
- Scalability
- Security
- Stability
- Testability
- Usability

Let’s analyze these quality attributes one by one.

6.1.1 Adaptability

All the different applications that want to use the SOCH API will have to use a REST API over HTTP. This is one of the most general and common ways to connect to software infrastructures today. Applications with other requirements could always build a proxy/façade pattern solution [Buschmann] for the SOCH API.

All the different content providers will have to use a bridge pattern [Buschmann] component in order to translate the local database to the semantic web SOCH RDF format. Templates for Windows and Java environments are available and used. No other environments have been requested, but could be built if needed.

6.1.2 Availability

Most of today's web applications have some requirements on availability, and the SOCH infrastructure is no exception. There are no serious implications if the services are down occasionally, but it is important for reasons of good-will and trust to provide a high level of availability.

All the SNHB web applications are reached through two web fronts, so there is redundancy on that level. The SOCH infrastructure also keeps on working if the database (repository) is unavailable. This is achieved by using a Java Solr index and having all the RDF structures included in the index. The repository is actually only used for storing new or changed objects after harvesting, and to build or rebuild the Solr index.

This leaves the main SOCH component as the only single-point-of-failure (SPOF). To avoid this SPOF, we can make use of patterns like Replicated Component Group and Master-Slave [Buschmann] with some enhancements. The solution is to duplicate the SOCH component, but only use one indexing engine. Build and optimize the Solr index in one of the SOCH nodes and copy the index to the other SOCH node. Use a heartbeat mechanism to decide if both or only one of the nodes are/is available for client requests. (As of autumn 2012 this functionality is partly developed and tested but not yet implemented, see figure 14.)

6.1.3 Extensibility

The SOCH architecture is developed with the concept of “loose coupling” at heart. The components are well described with a clear separation of concerns. The main component consists of two sub-components: the SOCH engine and the Solr index. The Solr index can be replaced by another indexing software if necessary. That is not a small operation, but it is possible. As of 2012, it is difficult to find a scenario where the Solr open source code would not be sufficient or possible to extend in-house.

The SOCH engine consists of the following packages:

- api – this package includes a dozen API methods, some of which have been quickly developed by customer request. There are abstract classes to build from and new methods can be developed with a minimum of effort.
- apikey – this small package handles API keys for customers using the SOCH API (see chapter 4 for a description of the SOCH API).
- harvest – the whole harvesting mechanism is included here. It builds on open source for OAI-PMH (Open Archives Initiative Protocol for Metadata Harvesting). It could be replaced by another harvesting mechanism with some effort. As of 2012 it is very difficult to find a scenario where this transport mechanism would not be fit for use.
- index – support code for the Solr index sub-component. This code would have to be replaced if Solr were to be replaced.
- manipulator – support package for Google indexing.
- organization – content providers can manipulate their own organizational metadata: this package handles that interface.
- resolve – this package makes sure that the URIs and URLs work as stipulated, i.e. that RDF, XML or institution website redirects are returned to the user or application making the request.
- sitemap – also concerned with Google indexing.
- spatial – support for spatial needs and different databases (Oracle and Postgres).
- statistic – handles statistics of user and application uses of the SOCH API.
- util – some general utilities used by other packages above.

The above packages should be easy to extend and new packages are reasonably easy to append with the loose coupling and separation of concerns implemented so far in SOCH.

6.1.4 Interoperability

Interoperability is achieved by providing templates for both Microsoft (.Net) and Java environments that can be used by content providers. Other platforms could also be supported at request, but this has not been addressed so far. At the beginning of the SOCH development there were requests for local ports based on Delphi, but these institutions migrated to Java before SNHB begun developing Delphi support.

Applications using the SOCH API have to use the HTTP protocol and can thus use any of the platforms supporting HTTP communication.

6.1.5 Maintainability

Most of what is said for “Extensibility” above is valid here as well. One area however, where maintenance could be simplified is when it comes to testing of new content provider material. The tools and methods for checking XML and RDF structures, as well as the SOCH protocol adherence, could and should be improved. This is also a usability aspect.

6.1.6 Modifiability

For the SOCH infrastructure, modifiability is more or less a subset of “Extensibility” (specified above). Some of the open source packages are not documented well, which means that developers tend to solve challenges outside of these packages when the most proper place would be to solve them inside those packages. This doesn’t happen very often, but must be documented properly when it occurs.

6.1.7 Performance

Users and applications utilizing the SOCH API should have response times good enough not to think of this as an issue. When using e.g. Kringla, both the presentation layer (Kringla) and the infrastructure (SOCH) add up to the processing time. Responses in Kringla are most often delivered within the second, and there are no signs of users complaining about performance.

Furthermore, the number of semantic objects in SOCH will not grow very fast. SNHB has already made some of the largest cultural heritage databases available, and SOCH includes more than four million objects as of 2012. If the number of objects would be eight million by 2018 this would be a great and unexpected achievement. Although not tested properly, this doubling of objects would probably not raise any performance issues even with the hardware currently in place.

New demands for services could of course raise new performance challenges, and should be designed with that in mind.

Besides the obvious concurrency of handled client requests, the following design choices have contributed to the high performance so far developed:

- The repository database is not needed at runtime. Since SOCH uses a Solr index for searching and other API demands, the full RDF structure for each semantic object has also been included in the Solr index. The repository is only used when harvesting (updating the objects) and when rebuilding or optimizing the index.

- Harvesting of many services can be carried out simultaneously. It is only in the indexing phase that services must be handled one at a time.

The change suggested for “Availability” (improved redundancy) would also enhance performance.

6.1.8 Portability

SOCH was from the start supposed to be portable, in case the government wants to move the responsibility to an institution other than SNHB. SOCH is indeed portable within the day, and the following is required at the receiving institution:

- a Linux environment (preferably CentOS)
- Apache Tomcat web server
- a relational database (preferably Postgres)
- Java development environment (J2SE) for continued maintenance and development. No specific framework is used (pure Java stack environment)
- DNS work is needed for remapping www.kulturarvsdata.se and www.ksamsok.se to new IP addresses if so decided
- some of the content providers may have to open their firewalls for the new SOCH harvester IP address

6.1.9 Reusability

SOCH is a specific service on a small market. There is no big demand of components. SOCH itself uses a lot of open source Java packages though.

6.1.10 Scalability

The need for scalability in the data layer (harvesting) is not likely to increase (see "Performance" above). If the number of applications increases quickly and/or the traffic increases from the average application, the SOCH infrastructure could and should be duplicated in order to achieve reasonable availability. This track of reasoning is covered by the discussion about "Availability" above.

6.1.11 Security

SOCH is only exposed through the well-defined SOCH API. Security risks are therefore smaller than web applications in general. For content providers, however, there is a login functionality used to update and maintain organization specific information. If a password like that gets exposed, there is a risk of an intruder messing up the information for that content provider. The information can be easily restored after such an incident.

6.1.12 Stability

For SOCH (and possibly many other systems) "Stability" can be seen as a subset of the "Availability" quality attribute.

6.1.13 Testability

SNHB does not maintain a solid test organization. In the case of SOCH, tests can be done on test machines that are more or less exact copies of the production machine. One difference is that the test machines are connected to a test repository, not to the production repository. This testability is considered enough as of 2012, and difficult to improve upon with the current test organization.

6.1.14 Usability

Usability is important for applications to accept the SOCH API. So far the API has got very good feedback from applications and hackathon events.

However, the documentation of the API could be further improved. And the support for new content providers validating their mapping against the SOCH protocol should be enhanced considerably.

6.2 Feedback and Determination of Challenge

A poll on the importance of the attributes listed above was carried out by the main crew working with SOCH (Lars Lundqvist, Johan Carlström and Henrik Summanen from the business side, and Hanna Glansholm, system developer) Following is a summary of the scores:

- Availability 4
- Performance 3
- Usability 3
- Modifiability 1
- Scalability 1

This report will therefore focus on the availability (and performance) architecture enhancement of duplicating the SOCH component using only one master index.

6.3 *Proposed Solution to Eliminate the Challenge*

Set up a second SOCH server. Make it a secondary server that uses a copy of the index from the primary server. Remove harvesting capabilities from the secondary server. Implement failover in the Apache web fronts. Use one master index and replicate it from the primary to the secondary server.

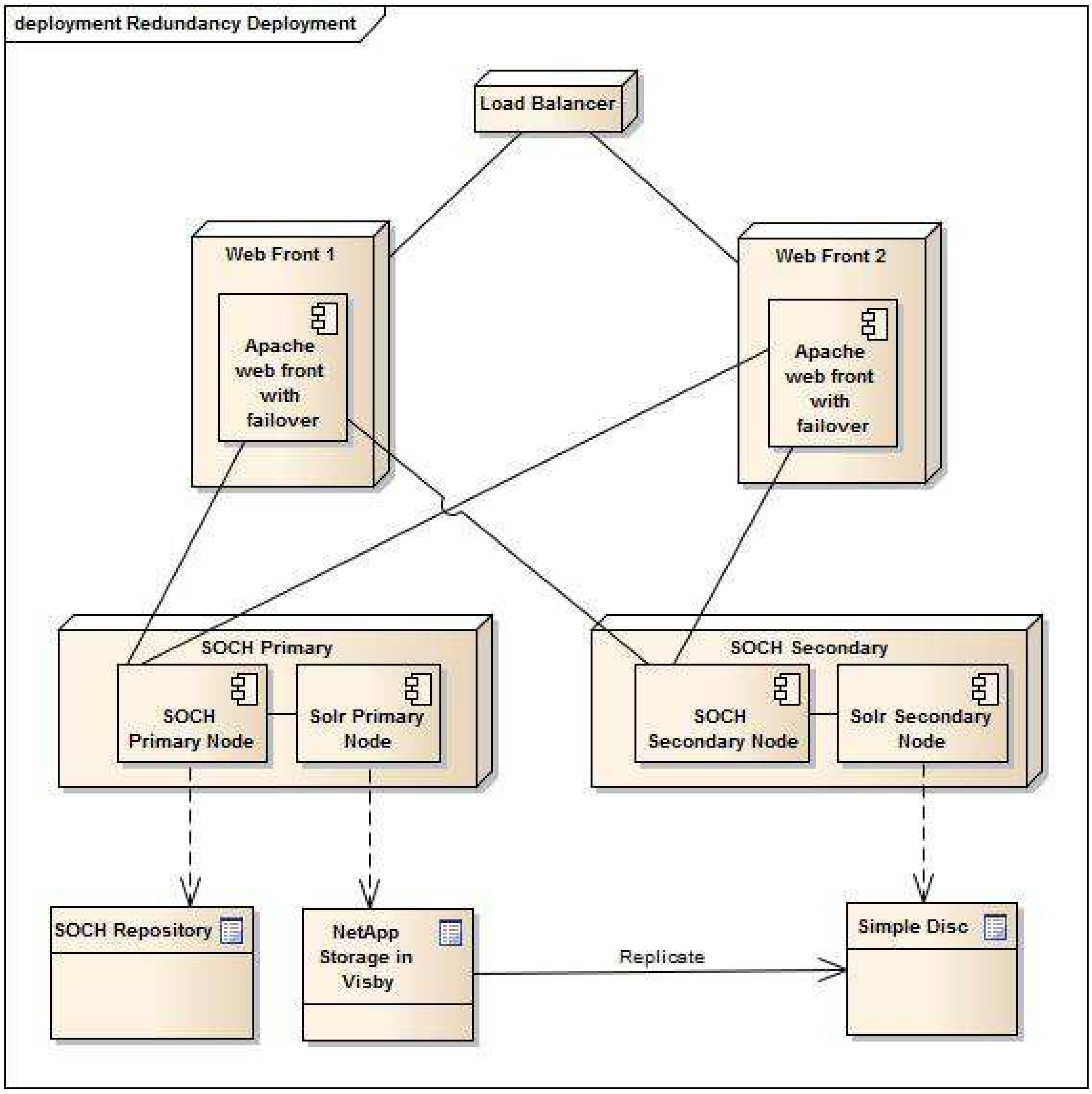

Figure 14: SOCH Redundancy Deployment

The load balancer uses round-robin to send traffic to the different web fronts. If a web front is unavailable for any reason, it is automatically removed from the round-robin queue. Actually, as of November 2012, a third web front is added for more redundancy and better performance.

7 Conclusions and Recommendations

The problem identified in the beginning of this report was to search and process cultural heritage information, residing in a long range of institutions. With the SOCH API in place, a search for urns found in Skåne would be specified as follows:

http://kulturarvsdata.se/ksamsok/api?stylesheet=&x-api=test&method=search&hitsPerPage=250&query=item= "urna"+and+place="skåne"

This will return objects from four different institutions.

SOCH is a well-functioning infrastructure, but has lacked the availability quality of redundancy. The business side of SNHB requested more redundancy and this has been developed without great effort. The solution is to add a second SOCH server with failover functionality. This solution is designed and tested successfully, but yet not implemented.

A further recommendation for SOCH is to improve the tools and methods for checking XML and RDF structures, as well as the SOCH protocol adherence.

8 Discussion

I chose to describe the SOCH infrastructure and identify the most important non-functional quality attribute to implement. I had great support from colleagues at SNHB for this work. I’ve learned a lot about architecture work that will come in handy in projects to come.

The redundancy solution outlined in this report will in fact be implemented in January 2013.

It is not clear in what ways SOCH will be developed further, but it is today an important player in the cultural heritage sector.

9 References

[Bass]	Len Bass, Paul Clements, Rick Kazman: Software Architecture in Practice, Second Edition, Addison-Wesley, 2011
[Buschmann]	Frank Buschmann, Kevlin Henney, Douglas C. Schmidt: Pattern-Oriented Software Architecture, Volume 4, John Wiley & Sons, Ltd, 2011

10 Appendix A

The W3 RDF Graph Generation in Figure 6 is attached as a PNG file.

11 Glossary

BBR – Bebyggelseregistret (BBR) is a national registry holding information about the built heritage

Content Provider – an institution, organization or other association delivering cultural heritage data and metadata to SOCH. This includes SNHB (cultural buildings, ancient monuments and cultural photographs), all different kinds of museums, local heritage institutions and community centers. Archives and libraries with digital resources (objects) are also included.

End user – a person working with research relating to the cultural heritage or a private person who is interested in the field, using Internet to find information

Europeana – a web site and infrastructure that resembles SOCH and has been influenced by SOCH. See http://www.europeana.eu/portal/

FMIS – the Swedish National Heritage Board's database for archaeological sites and monuments provides information about ancient and historical remains located on both land and under water. You will find information about remains such as rune stones, rock carvings, cemeteries, mines, remains of crofts, places of execution, shipwrecks and much more. The chronological span is wide **-** from the Early Stone Age to the 20th century AD.

Fornsök – the presentation layer application of FMIS

K-samsök – the Swedish name for SOCH

KMB – Kulturmiljöbild is the Swedish National Heritage Board's photographic database

Kringla – kringla.nu is a reference implementation which, so far, is the main presentation layer for the SOCH infrastructure

Kulturarvsdata – kulturarvsdata.se is the main entrance point to documentation of SOCH, but also the base URL for all the semantic material and persistent cultural heritage object URIs**.**

LIBRIS – a national search service providing information on titles held by Swedish university and research libraries, as well as about twenty public libraries. Here you can find books, periodicals, articles, maps, posters, printed music, electronic resources, etc. The LIBRIS Department of the National Library of Sweden is responsible for the operation and development of this online search service.

Linked Data - http://en.wikipedia.org/wiki/Linked_data

LOD – Linking Open Data, see "Linked Data" and "Semantic Web"

OAI -PMH – the Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH) is a low-barrier mechanism for repository interoperability. Data Providers are repositories that expose structured metadata via OAI-PMH. Service Providers then make OAI-PMH service requests to harvest that metadata. OAI-PMH is a set of six verbs or services that are invoked within HTTP. http://www.openarchives.org/OAI/openarchivesprotocol.html#Introduction

Provider – see Content Provider

RDF – Resource Description Framework, see http://en.wikipedia.org/wiki/Resource_Description_Framework

Semantic web – see http://en.wikipedia.org/wiki/Semantic_Web and http://upload.wikimedia.org/wikipedia/commons/0/02/Linking_Open_Data_cloud_diagram_2011-09-19.svg which shows the “Web of Data”. Note that SOCH is part of it, far out to the right in the diagram.

SOCH – Swedish Open Cultural Heritage is the software infrastructure described in this report

Solr – Solr is an open-source Java package for indexing and searching structured XML material and more. See http://en.wikipedia.org/wiki/Apache_Solr.

SNHB – Swedish National Heritage Board is the agency of the Swedish government that is responsible for heritage and historic environment issues

SPARQL – http://en.wikipedia.org/wiki/SPARQL

Web 3.0 – http://en.wikipedia.org/wiki/Web_2.0#Web_3.0

Web of Data – see “Semantic Web”

XMPP – see http://en.wikipedia.org/wiki/Extensible_Messaging_and_Presence_Protocol