

OpenInfobutton Workflow Documentation

Table of Contents

[1 Architecture 1](#_Toc351980931)

[2 Processing Logic 2](#_Toc351980932)

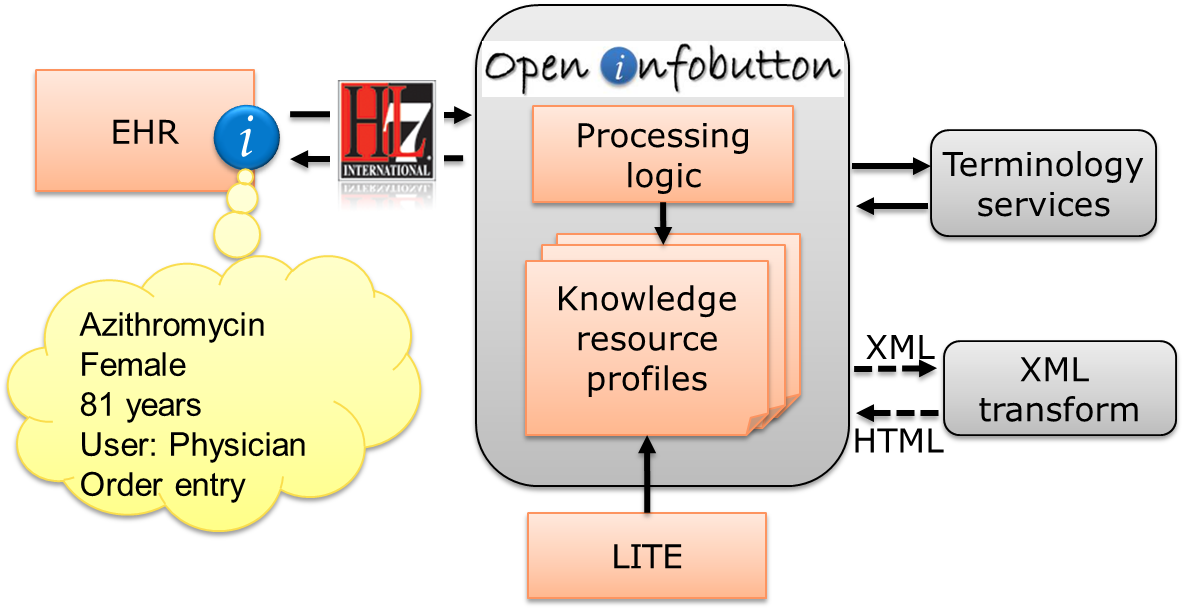
[2.1 Core Logic 2](#_Toc351980933)

[2.1.1 Context Matching 2](#_Toc351980934)

[2.1.2 URL Generation 2](#_Toc351980935)

[2.2 External Resources Handler. 3](#_Toc351980936)

# Architecture



# Overall Processing Logic

When a knowledge request is sent to OpenInfobutton, an algorithm is executed in two steps: context matching and URL generation. First, the logic selects the set of resource profiles that are configured for the sender. Next, the logic attempts to match the incoming context to one of the contexts defined in each of the selected resources. For the matched resources, the logic creates a set of URLs in the format of the target resource API. Each of these URLs corresponds to one relevant content topic that is covered by a resource. The final output contains the set of matched resources with URLs that retrieve content on a particular topic. These URLs consist of a set of context parameters for a resource search engine to process or a static URL that points to a particular document.

## Core Logic

The **infobutton-service** module is the main module that coordinates the workflow. It is a dynamic web project (2.5) configured with Spring-3.0 and it has the servlet mappings in web.xml.

There are 3 spring configuration files(core-data-annotation-context.xml,core-data-datasource-context.xml,core-profile-datasource-context.xml) that contain information about beans and contexts. These files are the first ones to be invoked once the project is launched.

**serviceParams.properties** is the properties file that has the necessary configurables used within the project.

**KnowledgeRequestSevlet.java** is the servlet which gets the http request and finally displays the response in the required format. The input infobutton URL is first transformed into **KnowledgeRequest** which holds the passed request parameters in a well-defined format of **CodedContextElements**.Then, the KnowledgeRequest is converted into REDS\_MT010001UV.KnowledgeRequestNotification complex type and the xml instance is written to the logs in OpenInfobutton’s MySQL database. After the log for the request is written the control is transferred to **KnowledgeRequestEngine**.

### Context Matching

**Overall, context matching is done in a sequence of steps. On each step a parameter of the knowledge request is matched against the resource profile CodedContextElement. If at any point of the process an incoming parameter does not match its CodedContextElement, the resource is considered not to be a match for the given context and the resource is not selected for the output.**

**More specifically, context matching works as follows:**

**1) KnowledgeRequestEngine** takes the **KnowledgeRequest** and does the Access Check and Task Check;

**2)** The **ContextProfileHandler** returns the contexts from the Resource Profiles after they are matched against the various CodedContextElements.

3) The Matching follows a fixed order and if the matching fails anywhere in the order then that context will not be added to the result. The matching order is: TaskContextMatcher, PerformerMatcher, InformationRecipientMatcher, EncounterMatcher, PatientContextMatcher, MainSearchCriteriaMatcher.

The **TerminologyInference** comes into play only when the MainSearchCriteriaMatcher is invoked. Based on the request and the profile, the following terminology inferences may be executed:

1) If the input code system is not supported by the resource, it can be transformed into a code system supported by the resource profile.

2) If free text is received instead of a code, the free text term is matched to a code in a code system supported by the resource according to its profile.

3) When the matching domain of a resource is configured to include all the descandants of a particular code in a hierarchy (e.g., all descendants of diabetes mellitus type 2 ICD10 code E11), a terminology inference is executed to fetch all the descendants of the specified code. This configuration is set in the includeDescendants attribute of each CodedContextElement.

To maximize efficiency all the codes generated by the TerminologyInference are stored in the KnowledgeRequest so that codes can be used for another profile if necessary without going through the entire process again and again.

### URL Generation

The KnowledgeRequestEngine gets the matched results from the ContextProfileHandler. This response includes all the resource contexts that matched the knowledge request. Next it calls the **ResponseGenerator** which returns the **AggregateKnowledgeResponse** according to the HL7 Infobutton SOA Implementation Guide.

The AggregateKnowledgeResponse consists of a set of resources (feeds) with one or more links (entries) to topics within each resource. To construct the AggregateKnowledgeResponse, the following process is executed:

1) A baseLink is constructed for each subTopic defined in the resource profile. Only the parameters that are set as search=true are included in the URL. Each link generates an entry in the AggregateKnowledgeResponse output.

2) If the resource is HL7URL compliant, the parameter names are pulled from the HL7 standard parameter names.

3) If the resource is not HL7URLcompliant, the parameter names are pulled from the resource profile.

4) If the resource is HL7KnowledgeResponse compliant (e.q., Medline Plus) then the API of that resource is called with the URL that was created, the HL7 knowledge response is parsed, and then added to OpenInofbutton’s AggregateKnowledgeResponse output.

The AggregateKnowledgeReponse output also contains metadata about the feeds and entries. Metadata are represented in category elements. The construction of the category elements follows the HL7 Infobutton SOA implementation guide. In summary, the category elements at the feed level reflect the parameters that were included in the knowledge request and have been sent to the resource. At each entry level, the category elements reflect only those parameters that have either been matched or searched for that link. In other words, only parts of the Knowledge Request that have been used by the profile, will be present in the category at the feed level.

Once all the results are processed, the AggregateKnowledgeResponse is returned to the KnowledgeRequestEngine. The xml is then constructed by marshaling the JAXBElements and a Document is returned to KnowledgeRequestServlet which then displays the result in XML or the request XSLT after xslt transformation.

## Terminology Inferences and the External Resources Handler.

This module is is decoupled from the core infobutton logic and is used for implementers to develop site-specific terminology inferences. The core OpenInfobutton project uses a combination of the UMLS Terminology Services (UTS), the RxNorm REST API, and an internal value set database. Other implementations may use other terminology servers such as Apelon DTS and 3M’s Health Data Dictionary.

The **infobutton-externalresources** module has a terminology inference interface and its implementations. The interface **ExternalResourceHandler** needs to be implemented in order for this module to work.

In the serviceParams.properties file the **service.MatchExternalSetClassName** has the name of the class which will be called for handling the external resources. This must be changed to the appropriate implementation-specific class at each OpenInfobutton site.

The methods within ExternalResourceHandler are :

/\*\*

\* Translates a code in one code system to an equivalent code in another code system.

\* **@param** code Code to be transformed

\* **@param** targetCS Target Code System

\* **@return** Code(Transformed Code) Returns a valid code if successful or null on failure

\*/

Code transformCode(Code code,String targetCS);

/\*\*

\* Retrieves a set of candidate codes from a set of code systems that approximately match a given term.

\* The search can be restricted to a subTree in a terminology (e.g., clinical findings in SNOMED-CT).

\* For now, the UTS implementation searches only in SNOMEDCT,ICD10CM and ICD9CM code systems

\* **@param** FreeText The Free Text term to be searched

\* **@return** List of Codes in the 3 code systems(if found) or an empty List on failure

\*/

ArrayList<Code> transformFreeText(String FreeText);

/\*\*

\* Checks if a code (code1) is a descendant of another code (code2).

\* **@param** code1 descendant

\* **@param** code2 parent

\* **@return** Boolean

\*/

**boolean** isDescendant(Code code1,Code code2);