OPAALS PROJECT

Contract n° IST-034824

WP10: Sustainable Research Community Building in the Open Knowledge Space

Del 10.16 – An automatic tagging tool integrated with OKS
+
Synchronisation in a community of OKS Appliances

Project funded by the European Community under the "Information Society Technology" Programme

Contract Number: IST-034824

Project Acronym: OPAALS

Deliverable N°: D10.16

Due date: M37(tagging tool – was done in time and integrated with RTS; this document has extra work being reported)

Delivery Date: September 2010

Short Description:

(1) An automatic tagging tool, developed and tested at IITK was integrated into RTS, a knowledge community aiming to foster share of solutions created by ordinary people to solve social problems, developed at IPTI.

(2) A synchronisation mechanism for a set of agropedia appliances has been designed and implemented.

Author: Rishi Kumar, Prateek Bhurat, Meeta Bagga

Partners contributed: IPTI

Made available to: Public

Versioning		
Version	**Date**	**Name, organization**

Quality check

Internal Reviewers: Saulo Barretto, Thomas Kurz

Dependencies:

Achievements*	See pages 5, 18
Work Packages	WP 10: Sustainable Research Community Building in the Open Knowledge Space
Partners	IPTI, FAO
Domains	Natural Language Processing, Information Retrieval, Operating Systems
Targets	System Administrators, IT Managers, Drupal Community
Publications*	Not as yet
PhD Students*	None
Outstanding features*	The experience with building the autotagger has further led to an automatic tagging tool for agriculture documents that was pursued separately with FAO.
Disciplinary domains of authors*	Rishi Kumar, IIT Kanpur, Computer Science Meeta Bagga, IIT Kanpur, Computer Science Prateek Bhurat, IIT Kanpur, Computer Science

The information marked with an asterisk (*) is provided in order to address Recommendation n. 4 from the Year 2 review report

Table of Contents

An automatic tagging tool integrated with OKS

Achievements

An automatic tagging tool has been integrated into the Guigoh platform. This tool can detect if the content is in Portuguese language and assign keywords automatically from the given ontology.

Outstanding features

The tool has a language detection feature and a learning component keyword assignment.

The word done is OPAALS was instrumental in subsequently developing an automatic tagging tool for agriculture documents. This toll is deployed at http://agropedialabs.iitk.ac.in/Tagger/ and has attracted considerable attention in the agriculture knowledge management community.

Objective

To develop and integrate of automatic tagging tool into the OKS version for RTS (Social Technology Network), a Brazilian knowledge community aiming to foster the share of social technologies created by ordinary people to solve social problems.

Background

Automatic Tagging Tool:

The basic task of a tagging tool is to extract keywords from a set of documents automatically. This task can be achieved in four ways keeping in mind the various use case scenarios available in the knowledge communities. These are as follows:

Approach	Available	Not available	Advantage(s)	Disadvantage(s)
1	• Training Set • Thesaurus	--	• Best accuracy • Highest Precision	• Processing time is huge. • Getting documents manually tagged is quite expensive. • Also constructing • Thesaurus is difficult.
2	Thesaurus	Training Set	• Cost of manual tagging is done away with. • Processing time is reduced as no training phase is there. • Tags are from controlled vocabulary	• Less Precision • Constructing thesaurus is difficult.
3	Training Set	Thesaurus	• Good Precision • Free Tags • The expense of constructing thesaurus is got rid of.	• Getting documents manually tagged is expensive. • Processing time is huge.
4	--	• Training Set • Thesaurus	• No training set required. • No thesaurus required. • No training phase	• Worst Accuracy

RTS

RTS stands for Rede deTecnologia Social, meaning Social Technology Network. It is an inititative by the Brazilian government to foster collaboration and sharing of social technologies, considered in Brazil as products, techniques and/or methodologies that are re-applicable, developed through a close interaction with a community and that represent effective solutions for social transformation. RTS works with twenty themes (water, energy, etc.) and feedback by people who used the technologies also enriches the system.

Tools Developed and Tested

We have developed an automatic tagging tool for Portuguese using the open source KEA [1] and tested it on documents in different languages like English, French and Spanish. The tool expects a knowledge model in the form of thesaurus in SKOS RDF format. Apart from this, a set of (say 100 documents) manually tagged documents is needed for training purpose. Also some linguistic knowledge is required - like stop words and stemming rules of the language. The process of automatic extraction of keywords is depicted below:

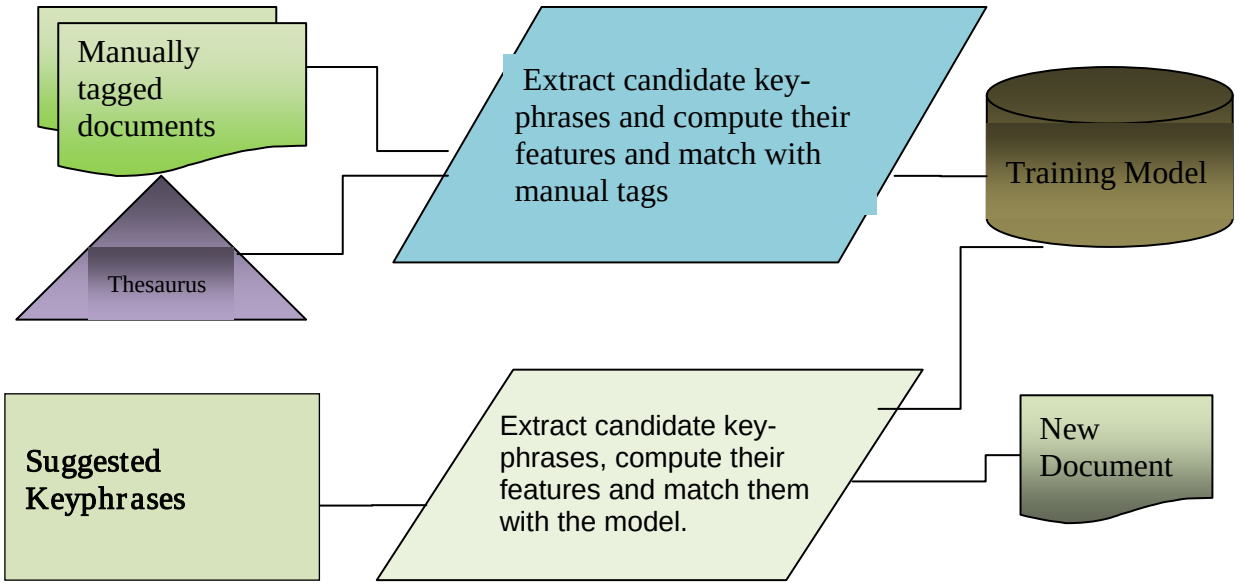

Figure 1. Workflow for the Autotagger

The tagging tool requires the following 3 items for its best output:

1) Knowledge model of the domain.
2) Some manually tagged documents for training.
3) Linguistic knowledge especially stop words and stemming rules.

To integrate the tagging tool into RTS we needed to adapt the tagger for Portuguese and this requires the building of the above three items for Portuguese. The idea is to demonstrate the

tagging tool in RTS for at least one theme and one experience. The keywords extracted by the tool will be given to the content publisher to choose tags from. Thus the tool will assist the members of RTS to assign tags to their documents, particularly text documents.

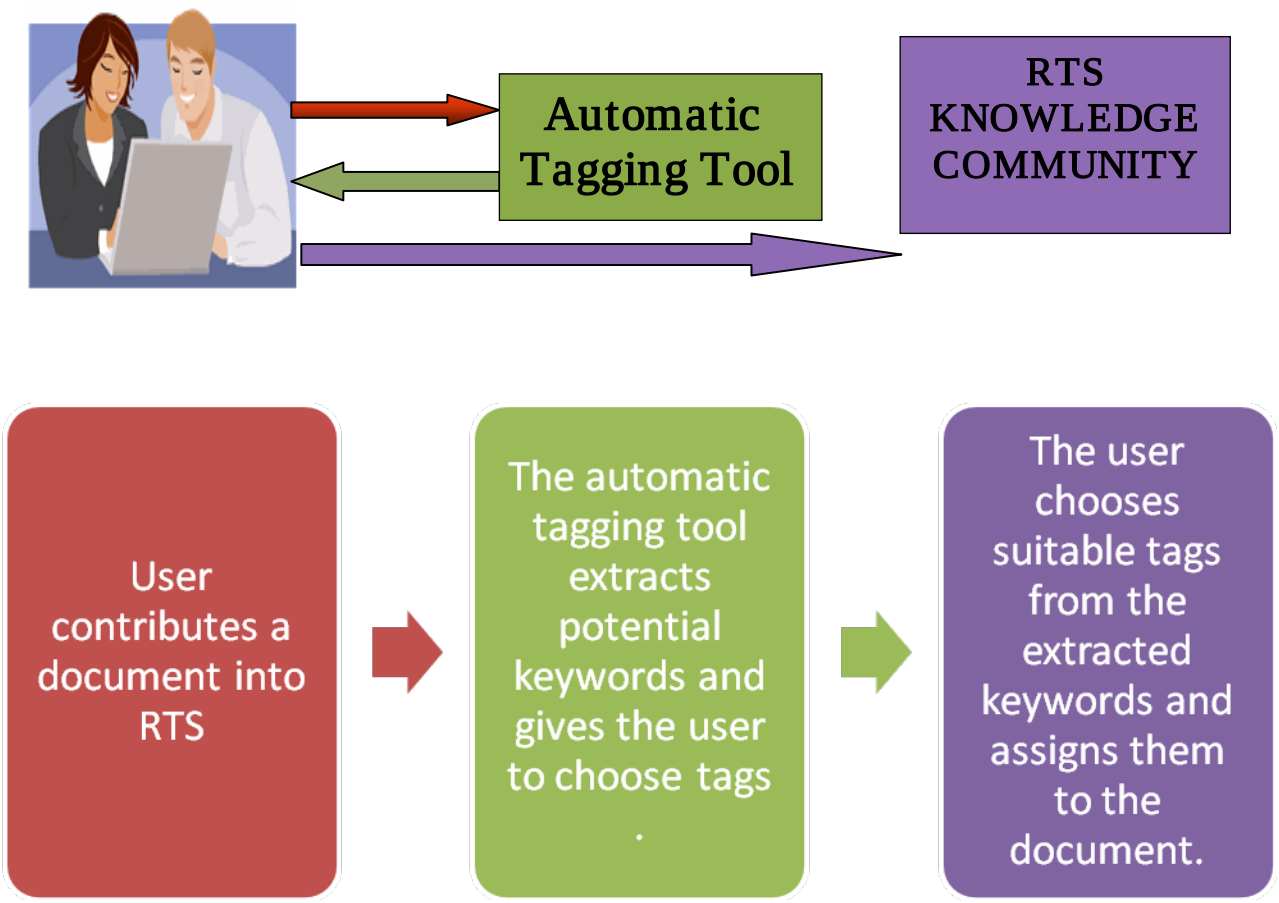

Figure 2. RTS community and the Automatic Tagging Tool

Tasks Accomplished

Internationalization of Tagger

a. Guessing the language(Potuguese or English) of the document given the text.

b. Tagging process adapted for Portuguese language(apart from English)
   a. Making iteratively the stopwords list for Portuguese.
   b. Stemming rules for portuguese implemented.

c. Tagger tested on one use case of RTS. Results are quite satisfactory.

Integration with RTS

d. Ran a separate servlet that calls the automatic tagger.

e. RTS calls the URL of the servlet and fetches the tags.

Ontology Building

f. Domain experts of RTS constructed the Ontolgy using Protégé.

Implementation Details

Language Detection

Several methods of feature extraction have been used for language classification, including unique tokens, frequent short word method, N-gram method, and ASCII vector of character sequences. Among the various approaches, we use the N-gram based application. NGramj[2] because of its easier implementation and comparable accuracy.

An ngram is a short sequence of bytes, characters or words. It has been observed that many properties of a underlying text (for example language, style, but even thematic focus) have a statistical stable impact on the ngram profile of this text. The ngram profile is the statistical distribution of ngrams, that is how often a certain ngram appears in a certain sequence. NGramJ only considers byte or character based ngrams. The reference n-gram profiles used by NgramJ had been built by TextCat[3].

Textcat uses text samples lying on the internet containing language markup.
ngrams are a rather classical instrument in Natural Language Processing (NLP) applications. *NGramJ* is a Java based library containing two types of ngram based applications. It's major focus is to provide robust and state of the art **language recognition** (or **language guessing** how some call it more correctly). Both types are meant to be embedded into larger applications.

NGramJ
This uses ngrams of **bytes** to determine from a sequence of bytes both language and encoding. In symbols:

NGramJ : byte[] --> (Language, Encoding)

CNGramJ
This uses ngrams of **characters** to determine the langauge of a character sequence. In symbols

CNgram : char[] --> Language

NLP for Portuguese

KEA has its keyword extraction algorithm implemented for three languages namely English, French and Spanish. The language specific components of the algorithm had to be patched for Portuguese. We had incrementally built the stopwords list in Portuguese beginning with

the standard Apache stopwords list. And also we have implemented stemming and term conflation algorithm for portuguese.

The processing of documents for the task of keyword extraction requires removing the stopwords and then stemming the n-grams.

a) Stopwords List

After literature review in research in Portuguese linguistics, we didn't find an acceptable set of stopwords to be used in the keyword extraction process. Therefore we built a new list of stopwords, justifying the selection of each stopword to ensure the credence of the results.

In the first step we built a exhaustive list with the following classes of words:
articles, pronouns, prepositions, conjunctions, adverbs, consonants, and vowels. Most of the words had been obtained from [4]. After the list was formed, it was noted that some words could also become nouns or adjectives, depending on the context. Then the ambiguities were sorted out by verifying the usage of all words in the Houaiss dictionary[4]. The most common classification of words was considered for the words classified in more than one category. The words found to be classified chiefly as adjective or nouns were kept out from the list.

The next step involved extracting keywords with the created stopword list and the output keywords which were not content bearing words were also put in the stopwords list. The resultant list has 262 distinct words.

b) Stemming

For highly inflected languages, such as Portuguese, stemming has even more significance. Here we implemented a suffix stripping algorithm, that is essentially a Potuguese version of porter stemmer, into the KEA system.

Algorithm 1 Stem(word)
Require: String word
1. word = processNasalidedVowels(word)
2. stem = word
3. r1 = findR(stem)
4. r2 = findR(r1)
5. rv = findRV(stem)
6. stem = step1(stem,r1,r2,rv)
7. if stem matches word then
8. stem = step2(stem,r1,r2,rv)
9. else
10. r1 = findR(stem)
11. r2 = findR(r1)
12. rv = findRV(stem)
13. end if
14. if stem matches word then
15. r1 = findR(stem)
16. r2 = findR(r1)
17. rv = findRV(stem)
18. stem = step3(stem,r1,r2,rv)
19. else
20. stem = step4(stem,r1,r2,rv)

```
21. end if
22. if stem matches word then
23. r1 = findR(stem)
24. r2 = findR(r1)
25. rv = findRV(stem)
26. end if
27. stem = step5(stem,r1,r2,rv)
28. stem = deprocessNasalidedVowels(stem)
29. return stem
```

Algorithm 2 Step1(word, r1, r2, rv)

```
Require: String word, String r1, String r2, String rv
1. for i=0; i <= suffix1:length - 1; i++ do
2. if r2.endsWith(suffix1[i]) then
3. return word.substring(0,word.length()-suffix1[i].length())
4. end if
5. end for
6. for i=0; i <=suffix2:length - 1; i++ do
7. if r2.endsWith(suffix2[i]) then
8. return word.substring(0,word.length()-suffix2[i].length())+"log"
9. end if
10. end for
11. for i=0; i <= suffix3:length - 1; i++ do
12. if r2.endsWith(suffix3[i]) then
13. return word.substring(0,word.length()-suffix3[i].length())+"u"
14. end if
15. end for
16. for i=0; i 6 suffix4:length □ 1; i++ do
17. if r2.endsWith(suffix4[i]) then
18. return word.substring(0,word.length()-suffix4[i].length())+"ente"
19. end if
20. end for
21. for i=0; i <= suffix5:length - 1; i++ do
22. if r1.endsWith(suffix5[i]) then
23. word = word.substring(0, word.length()-suffix5[i].length())
24. if word.endsWith("iv") AND r2.endsWith("iv"+suffix5[i]) then
25. word = word.substring(0,word.length()-2)
26. if word.endsWith("at") AND r2.endsWith("ativ"+suffix5[i])
then
27. word = word.substring(0, word.length()-2)
28. end if
29. else if word.endsWith("os") AND r2.endsWith("os"+suffix5[i])
then
30. word = word.substring(0,word.length()-2)
31. else if word.endsWith("ic") AND r2.endsWith("ic"+suffix5[i])
then
32. word = word.substring(0,word.length()-2)
33. else if st.endsWith("ad") AND r2.endsWith("ad"+suffix5[i])
then
34. word = word.substring(0,word.length()-2)
35. return word
36. end if
37. end if
38. end for
39. for i=0; i <= suffix6:length - 1; i++ do
40. if r2.endsWith(suffix6[i]) then
41. word = word.substring(0, word.length()-suffix6[i].length())
```

42. if word.endsWith("ante") AND r2.endsWith("ante"+suffix6[i])
then
43. word = word.substring(0,word.length()-4)
44. else if word.endsWith("avel") AND word.endsWith("avel"+suffix6[i])
then
45. word = word.substring(0,word.length()-4)
46. else if word.endsWith("_ivel") AND r2.endsWith("_ivel"+suffix6[i])
then
47. word = word.substring(0,word.length()-4)
48. end if
49. return word
50. end if
51. end for
51. for i=0; i <= suffix7:length - 1; i++ do
53. if r2.endsWith(suffix6[i]) then
54. word = word.substring(0, word.length()-suffix7[i].length())
55. if word.endsWith("abil") AND r2.endsWith("abil"+suffix7[i])
then
55. word = word.substring(0,word.length()-4)
57. else if word.endsWith("ic") AND word.endsWith("ic"+suffix7[i])
then
58. word = word.substring(0,word.length()-2)
59. else if word.endsWith("iv") AND r2.endsWith("iv"+suffix7[i])
then
60. word = word.substring(0,word.length()-2)
61. end if
62. return word
63. end if
64. end for
65. for i=0; i <= suffix8:length - 1; i++ do
66. if r2.endsWith(suffix8[i]) then
67. word = word.substring(0,word.length()-suffix8[i].length())
68. if word.endsWith("at") AND r2.endsWith("at"+suffix8[i]) then
69. word = word.substring(0,word.length()-2)
70. end if
71. return word
72. end if
73. end for
73. for i=0; i <= suffix9:length - 1; i++ do
74. if r2.endsWith(suffix9[i]) then
75. if word.endsWith("e" + suffix9[i]) then
76. word = word.substring(0,word.length()-suffix9[i].length())+"ir"
77. end if
78. return word
79. end if
80. end for
81. return word

Algorithm 3 Step2(word, r1, r2, rv)
Require: String word, String r1, String r2, String rv
1. for i=0; i <= suffixv:length - 1; i++ do
2. if rv.endsWith(suffixv[i]) then
3. return word.substring(0, word.length()-suffixv[i].length())
4. end if
5. end for
6. return word

Algorithm 4 Step3(word, r1, r2, rv)
Require: String word, String r1, String r2, String rv
1. if rv.endsWith("i") AND word.endsWith("ci") then
2. return word.substring(0, word.length()-1)
3. else
4. return word
5. end if

Algorithm 5 Step4(word, r1, r2, rv)
Require: String word, String r1, String r2, String rv
1. for i=0; i <= suffix.length - 1; i++ do
2. if rv.endsWith(suffixr[i]) then
3. return word.substring(0,word.length()-suffixr[i].length())
4. end if
5. end for
6. return word

Algorithm 6 Step5(word, r1, r2, rv)
Require: String word, String r1, String r2, String rv
1. for i=0; i <= suffix.length -1; i++ do
2. if rv.endsWith(suffixf[i]) then
3. word = word.substring(0,word.length()-suffixf[i].length())
4. if word.endsWith("gu") AND rv.endsWith("u"+su_xf[i]) then
5. word = word.substring(0,word.length()-1)
6. else if word.endsWith("ci") AND rv.endsWith("i"+su_xf[i]) then
7. word = word.substring(0,word.length()-1)
8. end if
9. return word
10. end if
11. end for
12. if word.endsWith("Ç") then
13. word = word.substring(0,word.length()-1)+"c"
14. end if
15. return word

Algorithm 7 findR(word)
Require: String word
1. for i=0; i <= word:length() - 1; i++ do
2. if vowels.contains(word.charAt(i)) then
3. if !vowels.contains(st.charAt(i+1)) then
4. return st.substring(i+2)
5. end if
6. end if
7. end for
8. return " "

Algorithm 8 findRV(word)
Require: String word
1. if word.length() _ 3 then
2. if !vowels.contains(st.charAt(1)) then
3. for i=2; i <= word:length() - 2; i++ do
4. if (vowels.contains(st.charAt(i)) then
5. return st.substring(i+1)
6. end if
7. end for
8. else if vowels.contains(st.charAt(0)) AND vowels.contains(st.charAt(1))
then
9. for i=2; i <= word:length() - 2; i++ do

```
10. if (vowels.contains(st.charAt(i)) then
11. return st.substring(i+1)
12. end if
13. end for
14. else
15. return st.substring(3)
16. end if
17. end if
18. return `` ''
```

Support for Non-plain text documents.

We also added a text filter for non-plain text documents. Apart from pdf format we also support for doc, docx, ppt, pptx, xls, xlsx using the openoffice service and javadocconverter.

```
$ s o f f i c e - headless - accept =”socket , port=8100; urp ; “
$ java -jar jodconverter-cli-2.2.2.jar <inputFi leName>.<ext>
<inputFi leName >. pdf
$ java -jar PDFTextParser.jar
<inputFi leName>.pdf  <inputFi leName>.txt
$
```

Integration with RTS

We integrated the tagger with RTS by deploying it as a servlet in the backend.

Results

We tested the accuracy of the Portuguese NLP component added into Kea. Though the testing set was small, the results were manually verified and were judged to be satisfactory. A sample of automatically extracted tags is given in the following figure

Auto Tags (Manual)	Auto Tags w/o training	Auto tags with training
Reaplicação difusão construções coletivas tecnologias sociais transformação social fortalecimento organizações financiadoras desafios escala	Tecnologia Rede Social Tecnologias Sociais Sociais pessoas Tecnologia Social reaplicar desenvolvem compartilhar	Força coletiva Força compartilhar Rede TS RTS Larissa Larissa Barros Barros concretizar TSs milhares Prêmio reaplicar Tecnologias Sociais caminho financiamento cooperar participar difusão

Sequel

Based on the OPAALS experience with implementing the Portuguese tagger in RTS, we went on to separately develop an automatic tagging scheme for agricutlure documents. This work was done in collaboration with FAO Rome and ICRISAT Hyderabad. The agrotagger basically computes a specified number of words(phrases), in decreasing order or probability that can serve as keywords for a given document. These words(phrases) are chosen from Agrotags - a carefully designed set with tagging agriculture documents in mind. Most of the agrotags (except a dozen or so) are in Agrovoc[5].

Agrotagger is available as a web-service. It can be easily called from any system - supply a document and get the keywords in return. We have linked it with Dspace[6] and installed it in ICRISAT. Integration with Eprints is in progress.

Agrotags comes in many flavors. The 'vanilla' version can be seen here:

http://agropedia.iitk.ac.in/agro_tag/agro_tree.html

The version augmented with scientific names is here:

http://agropedia.iitk.ac.in/agrotags_version2/agro_tree.html

We also consider the whole of Agrovoc as agrotags as well (see below).

Agrotagger can be modified to take any taxonomy and pick the keywords from there (of course we need to do some off line engineering). You can see it in action here:

http://agropedialabs.iitk.ac.in/Tagger/

Here you can use all the three versions of agrotags(with scientific names, without scientific names, all of agrovoc) to tag a PDF document. Please note that the PDF doc should be *text* PDF(generated from some word processor) not an *image* PDF (that is a scanned images converted to PDF). You can also tag a plain text file.

For *image* PDF we have developed an automated work flow where we do an OCR on the image and then use agrotagger. As you can expect there is some loss of precision, but our preliminary results show we are at 80%+ plus. This should be useful for *retrospective* tagging. Has been tested for older documents at ICRISAT.
This work has generated considerable interest in the Agriculture Knowledge Mangement community.

References

[1] KEA, Keyphrase Extraction Algorithm http://nzdl.org/kea
[2] Ngramj http://ngramj.sourceforge.net/
[3] Textcat http://www.let.rug.nl/~vannoord/TextCat/
[4] Houaiss Portuguese Dictionary http://www.dicionariohouaiss.com.br/
[5] Agrovoc,a multilingual, structured and controlled vocabulary designed to cover the terminology of all subject fields in agriculture, forestry, fisheries, food and related domains http://aims.fao.org/website/AGROVOC-Thesaurus/sub
[6] DSpace, open source software enables to share content, http://www.dspace.org/
[7] EPrints, a platform to build document repositories, http://www.eprints.org/

Synchronisation in a community of OKS Appliances

Achievements

.
A synchronisation mechanism for multiple instances of agropedia has been developed. This allows agropedia instances which have forked, to synchronise as and when a connection is available. The mechanism works with incremental updates

Agropedia is an example of open knowledge space where large number users connect, access and update the knowledge repository. This connection happens over the Internet (http). However there are locations which have poor or no Internet connectivity. In such cases a local agropedia - an agropedia instance - is installed. This results in each of the agropedia instances evolving independently resulting in an inconsistent set of instances. We developed a simple algorithm to synchronise these multiple instances as and when a connection becomes available. We deal only with the problem of addition and not deletions.

Outstanding features

The synchronisation mechanism allows the full use of the appliance developed in D10.15. With this additional feature appliances can be deployed in locations with poor connectivity, evolved independently and brought to synchrony as and when a connection becomes available.

Introduction

Usage of Information and communication technology (ICT) in developing countries is still a challenge due to poor connectivity. A case in study is the agriculture knowledge portal called agropedia[1]. Agropedia has a lot of agriculture content which is semantically indexed and is expected to be accessed on line – with users also contributing to the content. However this asks for an online connection which many of the potential users do not have. To negotiate such circumstances we have come up with a stand-alone deployment option, an agropedia appliance. These are small handy devices designed in such a way that it can act as agropedia standalone server and can connect many systems to it, for places with low or no internet connectivity like the Krishi Vgyan Kendras or village knowledge centres. Appliance contains an optimised Linux, able to run agropedia website on top of the Drupal Content management system[2]. An Asus eee pc is the hardware base of the appliance.

Deliverable D10.15 gives a detailed description of the appliance[3].

Now here comes the question: if we have multiple instances of the same site running, how do we keep them in synchrony? This research is conducted to find an optimized solution to this problem.

After a number of experiments undertaken and also keeping in mind the low bandwidth issues of the installation point (Krishi Vigyan Kendras or Village Knowledge Centers), we created an incremental updates solution.

Incremental Updates Implementation

Agropedia is built on top of the Drupal content management system. There could be several ways of achieving our objective of having incremental updates over Drupal. We identified the need to work at a node level to support incremental updates due to the fact that node being the smallest entity in a Drupal hierarchy. Any post/page/story in Drupal is stored in the form of nodes. The files in Drupal are organized in the following hierarchy - refer Fig.1.

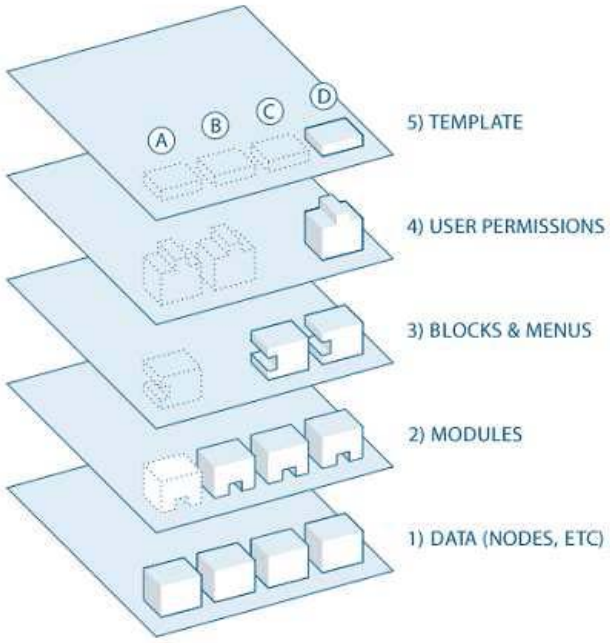

Fig. 1. Drupal Hierarchy

Algorithm for Incremental Updates

We now describe synchronisation algorithm with incremental updates. This algorithm caters to changes in any of the nodes and provides a mechanism to synchronise all of them with only incremental copies. This algorithm does not cater to deletes or changes in a node – it addresses only additions.

Assumptions

The parent node is the node which is copied into an appliance and deployed as a child node. There could be a large number of child nodes.

In each child node store

- the last checkpoint (up to this node number all updates have been sent to the parent server).
- folder containing updates to be sent to the parent server.

In the parent node store:

- the last checkpoint (up to this node number all updates have been sent to the child). It will be the same for all children (we will see how later).
- folder for each client containing set of updates to be communicated to the corresponding client.

We are dealing with only node insertions (deletions are not being taken into account as of now).

Example walkthrough the algorithm

Now, consider the case when a child gets updated. It looks up to find the last node checkpoint and creates sql queries for node updates after the checkpoint and writes them to folder from where they will be relayed to the parent. Refer Fig.2.

Server (Parent) 0-100	**Child 1 0-100**	**Child 2 0-100**	**Child 3 0-100**
Last Checkpoint			
-100	-100	-100	-100
Send_updates			
1 -	P -	P -	P -
2 -			
3 -			
Recvd_updates			
1 -	P -	P -	P -
2 -			
3 -			
Server (Parent) 0-100	**Child 1 0-120***	**Child 2 0-100**	**Child 3 0-130***
Last Checkpoint			
-100	-120	-100	-130
Send_updates			
1 -	P -100-120	P -	P -100-130
2 -			
3 -			
Recvd_updates			
1 -	P -	P -	P -
2 -			
3 -			
Server (Parent) 0-100	**Child 1 0-120**	**Child 2 0-100**	**Child 3 0-130**
Last Checkpoint			
-100	-120	-100	-130
Send_updates			
1 -	P -	P -	P -
2 -			
3 -			
Recvd_updates			
1 - 100-120	P -	P -	P -
2 -			
3 - 100-130			

Fig2. Child sending updates to parent

Consider the case when parent is updated:

Server (Parent) 0-150*	**Child 1 0-120**	**Child 2 0-100**	**Child 3 0-130**
Last Checkpoint			
-150	-120	-100	-130
Send_updates			
1 - 120-150	P -	P -	P -
2 - 100-150			
3 - 100-120			
Recvd_updates			
1 -	P -	P -	P -
2 -			
3 -			
Server (Parent) 0-150	**Child 1 0-120**	**Child 2 0-100**	**Child 3 0-130**
Last Checkpoint			
-150	-120	-100	-130
Send_updates			
1 -	P -	P -	P -
2 -			
3 -			
Recvd_updates			
1 -	P - 120-150	P -100-150	P -100-120
2 -			
3 -			
Server (Parent) 0-100	**Child 1 0-150**	**Child 2 0-150**	**Child 3 0-150**
Last Checkpoint			
-150	-150	-150	-150
Send_updates			
1 -	P -	P -	P -
2 -			
3 -			
Recvd_updates			
1 -	P -	P -	P -
2 -			
3 -			

Fig.3 Parent server updating child server

Either it is updated at the main server itself or it has received updates from a child and hence has got updated the parent looks up the last checkpoint and creates the corresponding sql queries excepts for the child from which it received its updated and stores them in corresponding folders for each child. Form these folders the updates can be relayed to the corresponding child. The parent now reset the new checkpoint which is the last node number. When the child receives the updates it resets its checkpoint to the last node number. Refer Fig.3.

Note- If a set of updates is available for a child but cannot be relayed as the child is not up and another set of updates become available then these new set of updates are just enquired to the existing set of updates.

Correctness

A child node can never skip an update if it is accessible from the parent server. In case the child node is not accessible from the parent node, all it's updates are accumulated and transferred the next time the child node connects to the main server. In this manner we succeed in keeping the Appliance website instance synchronized with the main website with the least amount of resources and updates.

Deployment and Testing

The entire setup has been implemented using PHP scripts and ftp servers. Whenever a client appliance makes changes on its local server, a PHP script reads these changes from the database and converts them to SQL queries. These queries then need to be relayed to the main server. The last update id needs to be stored. A FTP server written in python sypFTP has been deployed on the devices which are used to relay and receive the updates in the mechanism discussed above. When the updates reach the main server, a PHP script modifies its database by performing the updates and sends these updates to the clients (except the one from whom it received) using FTP.

Conclusion

This simple synchronisation scheme allows each of the child nodes – appliances located in areas where the connectivity is sporadic – to evolve independently yet synchronise with each other and the parent as and when there is connection established. The scheme needs to address issues when there are deletions.

References

[1] agropedia http://agropedia.iitk.ac.in/
[2] Drupal http://drupal.org/
[3] A KM appliance – for deployment in an agricultural knowledge management setting, OPAALS Deliverable D10.15