

EDTC Feasibility Study Grant Report

PictureBox

This report details work carried out at Edinburgh University's School of Informatics in respect of a research grant awarded to Interface3 in May 2011. The individuals involved were:

- Kate Ho, of Interface3, who devised the idea for the project and secured the grant,
- Kate Byrne, of the School of Informatics, who carried out the work.

The grant of £5,000 paid for 12.5 days research effort (though in fact more time was spent on the project) leading to the creation of an application that is here dubbed *PictureBox*. This is merely a working title and may be changed by Interface3 when they deploy the application to their market.

1 Background

Interface3 is a young company that started as a spin-off from the School of Informatics and has been running for around two years, mainly doing consultancy work around applications for interactive tables and tablets in the education sector, particularly primary schools. They are in the process of expanding into product-based business, with software applications tailored for this sector. The grant from EDTC Technology Gateway was to enable research into developing a new product, and a working prototype application has been delivered as a result.

The project was specified by Interface3 with Edinburgh University as the research partner. As specified in the grant application the overall aim was to produce “an online tool for Primary School teachers to find and download relevant and usable images”.

As an example, a primary school teacher putting together a lesson on “Healthy Eating” typically needs to collect suitable images, say from Google Images. If “fruit” is to be illustrated this means either searching for the generic term and then painstakingly editing results to extract examples of specific fruits, or else thinking up a list of sub-types and searching for each one in turn. Whichever route is taken it is a time-consuming chore that Interface3 hope to automate for them. Continuing with the “fruit” example, the objective can be succinctly explained by the picture included in the original grant application, shown in Fig. 1.

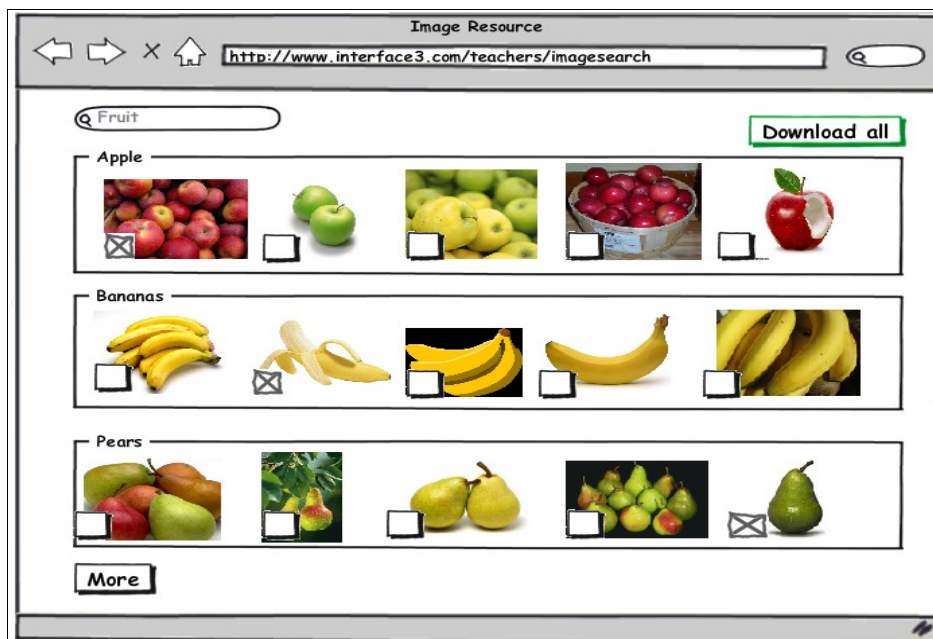


Figure 1: The original application proposal

As Fig. 1 illustrates, the objective was to devise a way of allowing the user to enter a generic search term and have image results returned that are clustered into instances of sub-categories of the generic term. When the results are presented the user should have a simple way of choosing the images they want to keep and saving these in a single operation.

As the proposal document explained, the difficulty in achieving this deceptively simple result is in generating the sub-category terms that allow clustered results – in this illustration, turning “fruit” into “Apple, Banana, Pears...”. Constructing pre-defined lists was rejected as a solution because the aim is to provide a flexible tool that can be used for any search term. Therefore a solution was required that involves natural language processing (NLP) to generate the clusters in real time. The preferred approach suggested in the proposal was to research and apply suitable methods for generating narrower terms given a broad term.

2 Project Architecture

There were two separate aspects to the project: the NLP research work and the practical business of developing a working software application. The NLP work is covered in Section 3, whilst this section describes the overall architecture of the system and the issues encountered.

In designing the system I tried to incorporate and balance the following objectives:

- an application that is very simple to use, requiring no instruction beyond a line or two of text on the screen presented to the user;
- fast query response for the end-user;
- platform independence (development was done on Unix but end-users will probably run Windows or MacOS);
- component software tools that are free, without licensing restrictions, and based on standard languages that can easily be maintained;
- rapid development, as time was extremely limited;
- operating in a web browser so that no software need be installed by the user;
- as far as possible keeping the functionality on the client side, to minimise the need for server maintenance.

After some research into what is currently available, the Google Web Toolkit (GWT) was chosen as the development platform. This allows the software to be written in Java (with which I happen to be familiar) and deployed in Javascript, which meets the platform objectives listed above. The GWT is designed to integrate very easily with the Eclipse tool, a very widely used development environment (also free), and this was a significant bonus. The GWT is made freely available by Google and has an extremely extensive backing network, of ancillary tools and community support. In fact a possible drawback is that it is subject to so much enhancement that the APIs tend to change rather quickly. Some of the interfaces used in this work have already moved to “deprecated” status, so would need upgrading if the application is used for more than the next few years.¹

Writing directly in Javascript might have made maintenance easier for the future – as it stands, amendments are simple if done through the GWT but quite awkward outside it. However, the priority was to find a tool that was quick to learn and easy to use. There wasn't time to learn the full range of Javascript tools that would be needed, whereas Java was both familiar and provided with an immense range of existing tools to incorporate. The only restriction is that, naturally enough, one cannot use Java routines that are impossible to convert to Javascript.

¹ Deprecation is a standard feature of software API schemes. As new, preferred features and methods are introduced, the older tools they replace are labelled as “deprecated” and will cease to be supported after a given period, usually a few years. The delay is to allow programmers to upgrade their code to use the new methods. A possible drawback of Google's very wide range of APIs is that the whole architecture seems to change quite often. However, there are obviously many advantages to using very highly developed, free and open software tools like Google's.

Although a fully client side application would have been the ideal, it is in fact impossible when the functionality requires downloading and saving groups of files to the user's own computer. A more experienced web developer would have realised this at once – what a browser can write to disk is extremely restricted, for security² – and I very soon discovered it. When the user selects images to save there has to be interaction with server side code. The simplest solution was chosen: the client tells the server which images are wanted, the server then bundles them up (as a zip archive for convenience) and presents the bundle back to the client. When presented with a zip archive the default behaviour on the client is that the browser will bring up a “Save as” window and invite the user to choose where to save the image bundle. If the local browser happens to have an add-on installed that can handle zip archives then the user will, for example, be able to open the archive directly. Either result is equally satisfactory for this system. The functionality was fairly straightforward, but learning how to implement client-server operations took a couple of days or so. The excellent documentation provided for GWT is a huge boon – in this case <http://code.google.com/webtoolkit/doc/latest/tutorial/clientserver.html> was the key guide. After exploring the options I chose the RPC (remote procedure call) framework.

In a client-server system the server code is entirely under the control of the system operator (Interface3 ultimately, in this case) and can be of any architecture desired, as the restrictions pertaining to web browsers do not apply. Here it was simplest to use Java code on the server. The java programs can be maintained very easily through the GWT if desired, alongside the original client Java, and of course there is no restriction on the Java routines that can be used server-side as they do not get translated to Javascript. Figure 2 depicts the system architecture (with acknowledgement to Google, on whose explanation and diagram it is based). When the user clicks the “save” button in the browser a “ZipService” is called to bundle the chosen files into a zip archive and present them back to the browser.

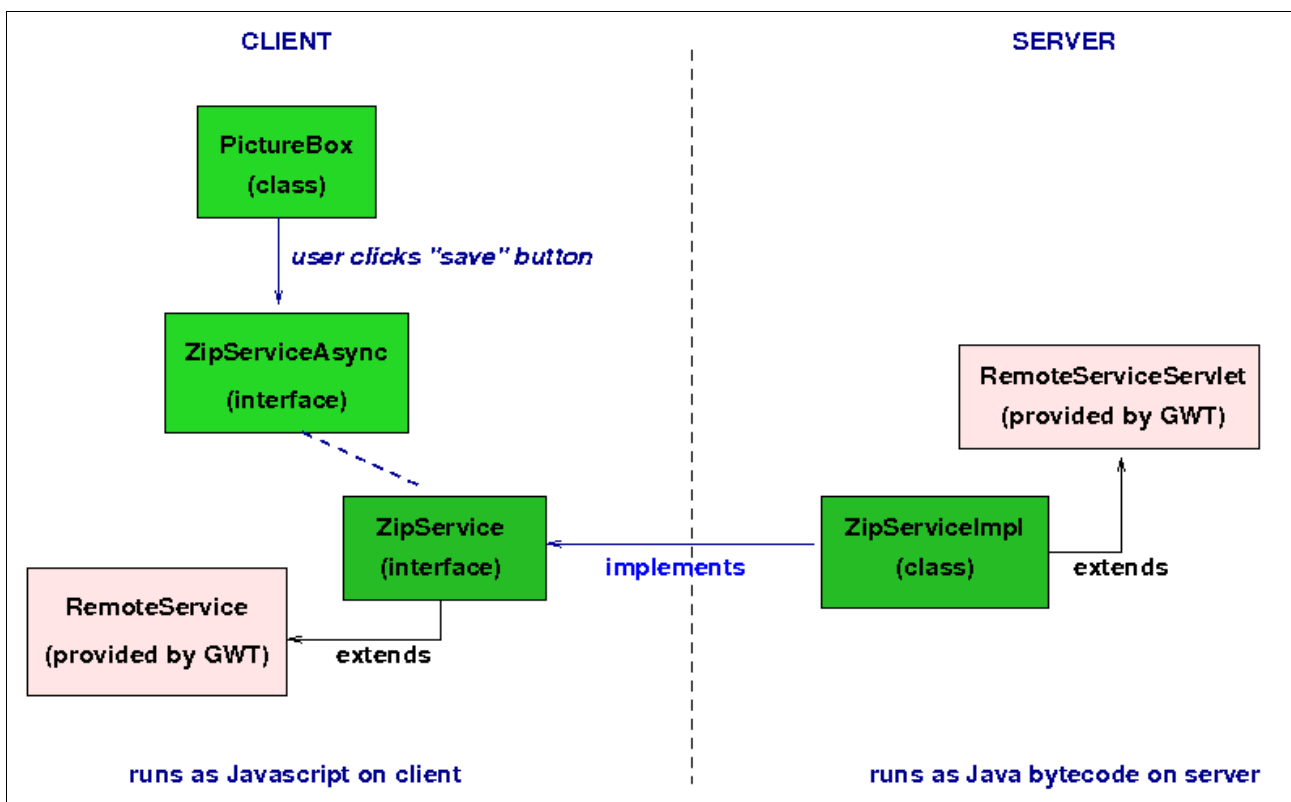


Figure 2: PictureBox system architecture

2 There is nothing to stop the user right-clicking on each individual image and saving them manually one by one, but having a button that saves a collection of images all at once is a very different thing, and the browser is not permitted to do it. If this were allowed it would be simple for a malicious web designer to place unwanted material (like a virus) on the user's disk by slipping it into the bundle.

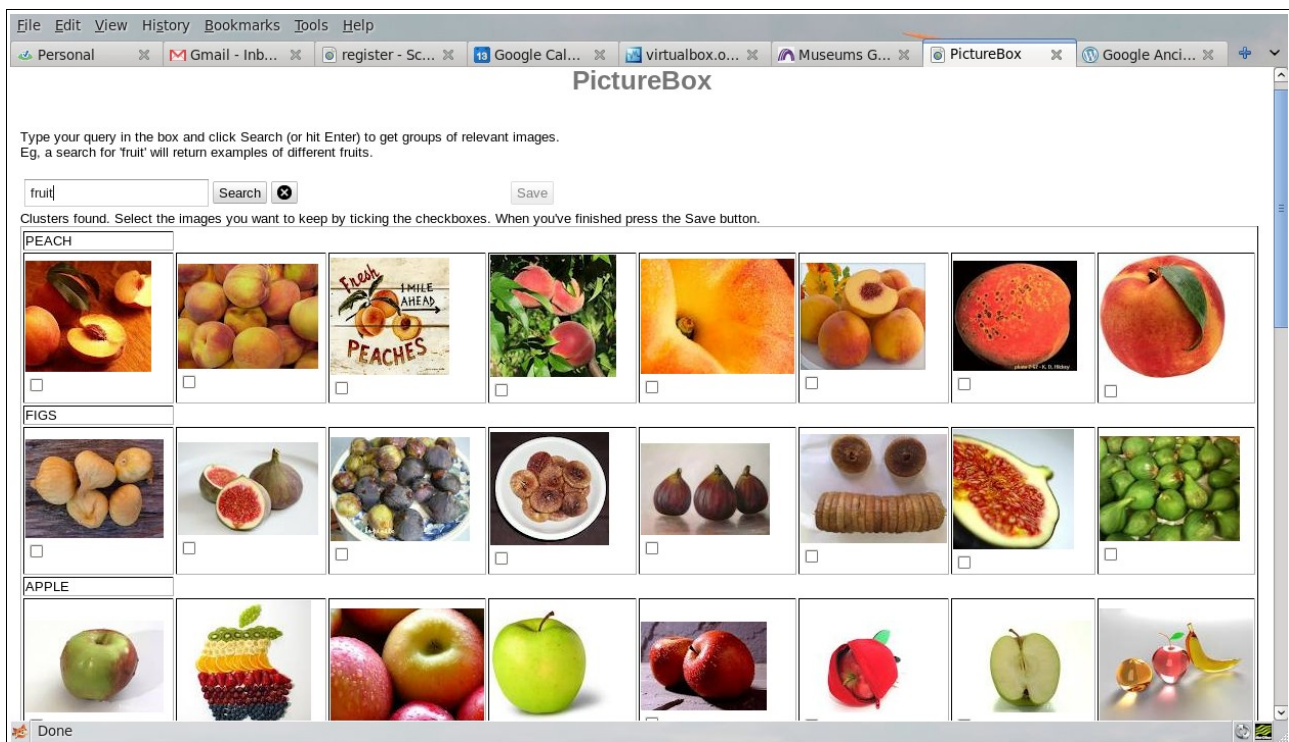


Figure 3: The final PictureBox application

Screen design and presentation were not part of the project brief so only the bare minimum was done in this respect. Figure 3 shows the final application as it appears in the user's browser when displaying search results – in this case for “fruit” as the query term. As the scroll bar on the right indicates, only the first few results are visible in this screen-shot. When the application is first loaded it presents an empty search box on a screen otherwise blank, apart from the one sentence guidance (illustrated in Fig. 3, “Type your query in the box...”) on usage. Mousing over the various buttons provides further very brief guidance, as “tooltips”.

The thumbnails are sourced by doing a Google Image search, which is an option available through the GWT. The “safe search” option was used, with value “strict”, bearing in mind the target audience is primary schools. For convenience in subsequent handling only images of a particular format are returned – all JPEGs, though any format could be chosen. The thumbnails come with their own bundle of properties which could be exploited, but in this case a deliberate decision was taken to provide only a mouseover tooltip giving the brief snippet of textual information that accompanies each thumbnail. Sections 5 and 6 discuss the reasoning behind this decision and possible extensions of functionality that could be added.

3 Clustering and Generation of Narrower Terms

The core of the project was the NLP aspect – solving the problem of how to generate labelled clusters of sub-types in response to a query for a generic term like “fruit”, “vegetables”, “US presidents”, “metaphysical poets”, or whatever. Some possible approaches were outlined in the original proposal and the first few days effort were spent in further research and simple experimentation to see what worked best in practice. Several different ways of tackling the problem were considered:

1. Treat it as an ontology generation problem. Given a term supplied by the user, assume that term to be at the head of a hierarchy and find the narrower terms one level below it. Then the narrower terms can be used for a series of separate web searches (using Google Images³) to return images to the user.

3 <http://images.google.com/>

2. Treat it as a clustering problem. Start with a web search for the user-supplied term and cluster the results, then label the clusters.
 - a) In theory the search could be for images and the clustering could use CBIR (content-based image retrieval) techniques. With this approach it might be possible to avoid multiple searches.
 - b) Alternatively one would use NLP clustering tools over documents found by a text search, and then run separate queries for images using the cluster labels determined by the system.
3. Use an existing term hierarchy or combination of such hierarchies. There are plenty of possibilities – Wordnet (Miller, 1995), dbPedia (Auer et al, 2007), YAGO (Suchanek et al, 2007), Cyc (Lenat, 1995), plus any number of domain specific thesauri such as the Getty ones for art and architecture (<http://www.getty.edu/research/tools/vocabularies/>) or Geonames (<http://www.geonames.org/>) for toponyms – each with different characteristics that would entail different system engineering.⁴
4. Start with a text search for the user-supplied term and use regular expression pattern matching (finite state automata) over the results to extract sets of narrower terms to use for a second round of searches for images.

All of these very different approaches have been used, with varying degrees of success, over the past decade or so of NLP research.

Automatic ontology generation is very difficult, even within a restricted domain such as fine art (Alani et al, 2003) or biochemistry (Blaschke and Valencia, 2003). Both of these examples are of complex systems that took months of effort to build. For the first, the Artequakt system, the ontology is only semi-automatically created, being edited by experts – clearly not a suitable method here. (It then forms part of a much wider knowledge base used for NLG (natural language generation) within a web-based information system on fine art.) In the second example a gene product ontology is created by first doing document clustering to generate the leaf nodes of a hierarchical tree and then working upwards towards the root, merging the two most similar nodes to generate the parent node at each stage. Clearly a bottom-up method is not applicable to our problem where, in effect, the root node must be the starting point, but the suggestion is that this is a more tractable approach than a top-down one.

The obvious conclusion was that the ontology generation approach is a non-starter for a short project like this, that had to produce a domain-agnostic system with fast search response time. However, the Blaschke and Valencia approach showed the possibilities in starting with clustering, so that was considered next.

Although a CBIR approach is an attractive idea – which Google are starting to provide practical tools for, with their image query by example prototype – it is too far out of the NLP scope to be considered seriously for *PictureBox*. My own experience in this field (albeit several years out of date now) is that image recognition is nowhere near the level that would be needed to make it practical for a system intended to deal with *any* search the user cares to enter. For example, it is just plausible that a CBIR system could correctly group apples and separate them from oranges (similar shapes but different colours), but there is no hope of it distinguishing, say, kestrels from buzzards (in a search for birds of prey), or at least not unless it were carefully trained for that specific domain. Presumably by releasing a public CBIR interface, Google are planning to build up a suitable database of searches precisely to enable a generalist system to be built, but even with their data gathering powers one imagines it will take several years.

Option 2. b) above, clustering over textual content, is a well-established technique. A two stage process would be needed: first cluster the text documents, then assign a suitable label to each

⁴ Using Google Sets (<http://labs.google.com/sets>) was also considered but quickly rejected. This tool, from Google Labs, attempts to detect the common factor in a set of related terms (such as “apple, banana, pear”) and will then search for more objects that fit the set (“orange, pineapple,...”). However, it requires seed terms – and if one has those the problem is already solved. Furthermore it seems to be a transient member of the Google Lab armoury, with no programmable API, uncertain licensing arrangements, and a frequently broken URL (ie it may have been withdrawn).

cluster. The second round of searches – for images – are then done using the label strings. The EU-funded SYNC3 project (<http://www.sync3.eu/>) uses the “cluster and label” method over news stories, to group together reports dealing with the same news event and assign a descriptive label to each story (Alex and Grover, 2010; Thuc and Renders 2011). This is an on-going project in which the Language Technology Group in Edinburgh’s School of Informatics is a partner, so it seemed worth investigating whether the methods could be adapted. It quickly became apparent that one would struggle to adapt the the highly complex SYNC3 software for *PictureBox*’s requirements in the time available. In any case the complexity of hosting such a system, which uses a high performance computing cluster to ensure processing speeds that can match news feed data volumes, ruled it out.

There are many other examples of untrained clustering systems⁵ in the literature but, as with the ontology generation systems, they tend to be highly complex and will only give reasonable performance if restricted to a specific domain.

The option of using a pre-defined terminological structure, such as some combination of the ones listed at item 3. above, was unattractive from the start, as it seems an unimaginative and inflexible approach. The limits of the final system would be defined by which thesauri or ontologies one chose to use and, as with all the approaches considered so far, it would be difficult to make the search responses really fast, unless the ontologies were copied locally. There might well be licensing restrictions for commercial use of copies of the data, and maintaining up to date copies becomes a tiresome overhead. Some practical experiments were done with WordNet and they showed that, far too often, the kind of query terms a teacher might want to use would not be present in the system.

After surveying these traditional but highly compute-intensive NLP methods, I started experimenting with a very much simpler approach, exploiting the feature of Google that allows one to search for matches on quoted strings. By combining the user-supplied search term with pre-defined patterns like “including”, “such as”, “comprises” and so forth, one can collect a set of results that will often include lists of sub-category terms. The term lists can then be extracted using regular expressions, with a high level of accuracy. This kind of approach, combining the use of fixed patterns with finite state post-processing, is common in the Information Retrieval literature and has been used for many years. For example Riloff and Lorenzen (1999) used simple “passive verb + trigger word” patterns to identify victims in texts about terrorist attacks; with trigger words such as “murder” or “kill” the pattern extracts texts snippets like “X has been murdered”, “Y was killed”.

A significant advantage of this pattern-based method is that one does not need to retrieve whole documents in the first, text-based, search. Since the snippet that a Google search returns will include the search string, when using the “quoted string” method, the list of terms to be extracted will almost always be within the snippet, which has the additional advantage of being a very small data item, easily handled in string variables in memory, so no saving of intermediate document results is required.

Extracting the term-lists is fairly straightforward though not completely error-free. A text snippet including the string “...fruit such as apples, bananas, oranges and pears.” is easy to deal with. A snippet including “...fruit such as these, or perhaps others.” will lead to errors as “these” and “perhaps others” are extracted as sub-category terms for the image search step. Many of the sub-category terms will be multi-word, eg personal names are usually two words but sometimes more (Edgar Allan Poe, etc), place-name strings can be many tokens long (The Peoples’ Republic of China), and so on. Therefore the patterns used to extract the term lists have to be something of a compromise and will make errors from time to time. I settled on regular expression patterns that would extract comma separated lists of terms comprising up to three tokens, with the final term being optionally separated by “and” or “or”, with or without an extra comma, optionally followed by “etc” and with trailing punctuation such as a full stop or semi-colon.

⁵ A trained system, where the cluster labels are pre-defined from a set of hand-labelled documents, will generally perform better than an untrained system – such as would be needed here – where the clusters must be determined without a fixed set of labels as clues, and the labels are then derived from the clusters themselves.

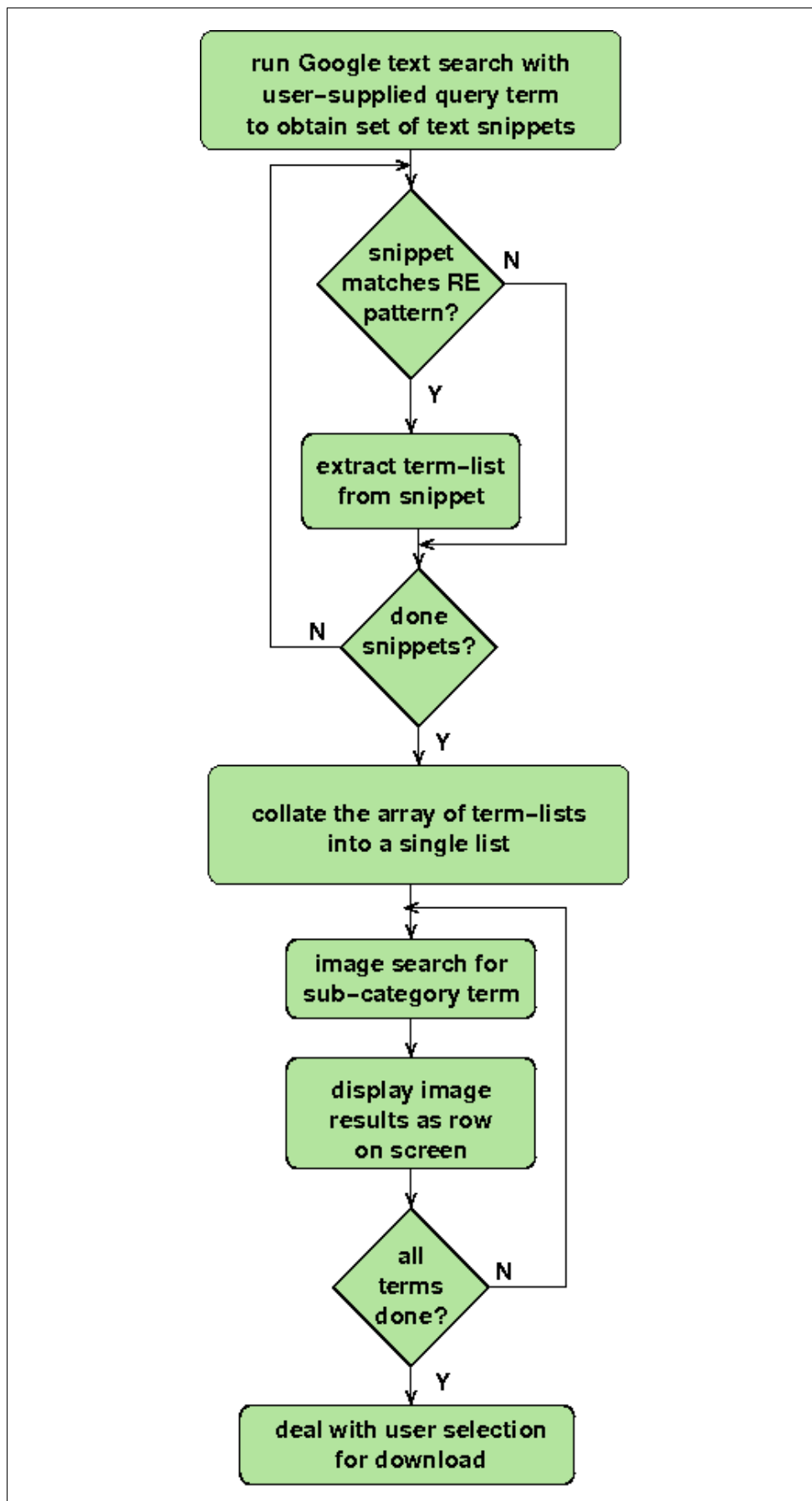


Figure 4: Processing flow in PictureBox application

By a process of trial and error the regular expression patterns were tuned until satisfactory results were being obtained in almost every search tried. As is discussed in Section 4 below, some of the searches tried in beta testing by Interface3 were of an unanticipated kind (eg “Superman”) and it is certain that the more formal ontological approaches explored initially would not have coped with them.

The method described will produce many overlapping sets of terms that need to be merged and deduplicated. The full processing pipeline is shown in Fig. 4. The final image searching is done in a loop, one search per sub-category term in the list. The original user query term is concatenated with the sub-category term to try to ensure relevance. For example, searching for “apple” without including “fruit” as part of the query will probably return images related to Apple Corporation. Section 6 discusses possible improvements that there was insufficient time to implement.

4 Using *PictureBox*

The application was extensively tested before hand-over to Interface3, and seems to perform well on a wide variety of query terms, from obvious “category” terms like “fruit”, “vegetables”, “birds”, “flowers” to more imaginative ones such as the “Superman” search tried in tests carried out by Interface3. The results for “Superman” are “invulnerability”, “heat vision”, “incredible strength” and “speed”, each with a set of illustrative images – which is as good as one could reasonably hope for, as it is hardly a “broad term” that one could expect to figure in a hierarchical ontology. Yet if one does search for this type of term, obtaining component characteristics is probably the desired result.

Because the method used is entirely agnostic about the kind of query entered, and relies on the power of Google’s search engine to find results, it provides an additional benefit for teachers in that it can be used as a convenient research tool. For example, the user does not need to know in advance that John Donne, Richard Crashaw, Francis Quarles and George Herbert were metaphysical poets; one simply searches for “metaphysical poets” and sees what is returned.

Figure 5 shows the result of searching for “birds of prey” (falcons, buzzards, golden eagles, the bald eagle, owls, eagles, condors kites). It is important to note that the results make no attempt

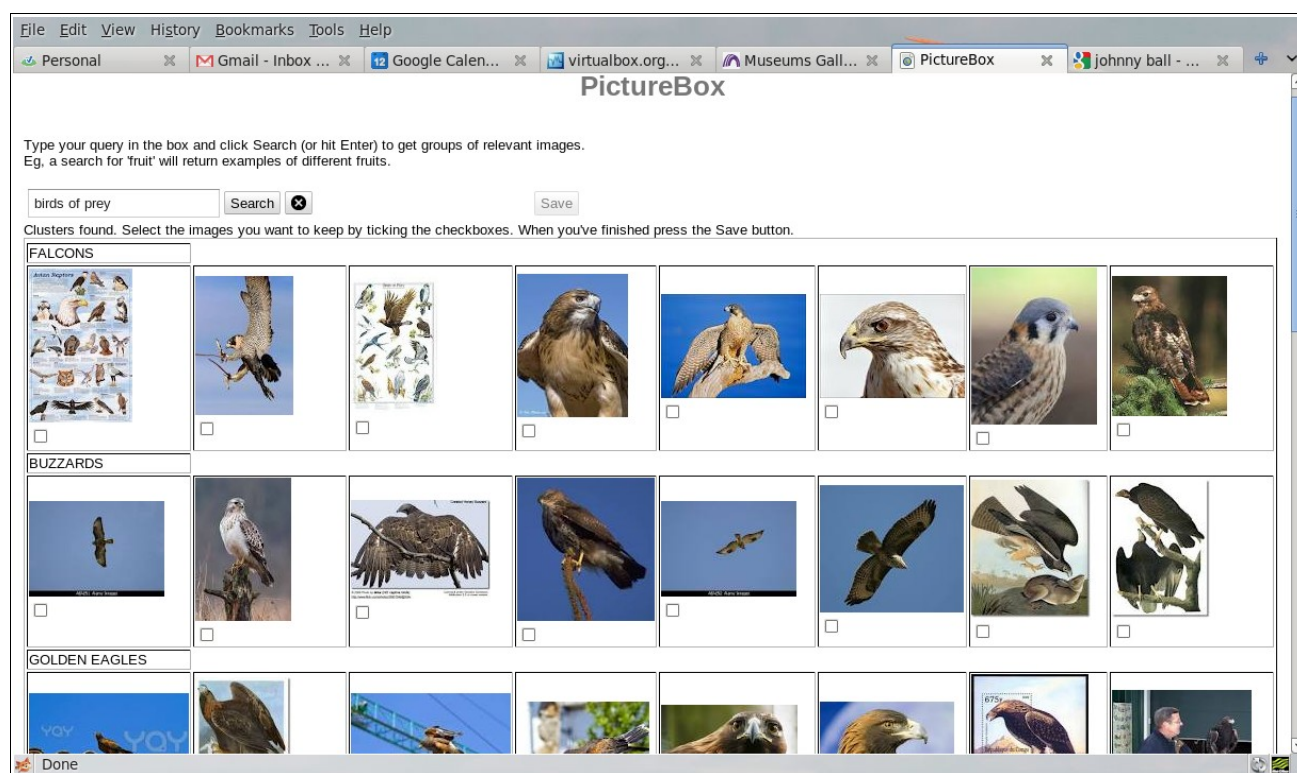


Figure 5: A search for “birds of prey”.

at being comprehensive. As in so many web search applications, where the number of results can easily be numbered in millions, the user is far more likely to be interested in precision (the results they see are correct) than in recall (they see all valid results). For the target audience of primary school teachers preparing lessons for children it seems reasonable to suggest that a sample of typical results is what's required. It has to be said that the results also have a certain whimsicality in that, just like any other Google search, they will change over time. Also like any other Google search, one sometimes has to reword a query a few times in order to get the desired results.

If no list of sub-categories could be extracted from the snippets returned by the original text search, the user gets a message saying that no image clusters could be found and they should try a different search. When images are returned, a one-line instruction on screen prompts the user to select the images to download, by clicking in the check-boxes under each thumbnail. Once the user starts ticking boxes, the "save" button is enabled and, when it is pressed, a confirmation message appears saying how many images have been selected and what the name of the file bundle will be; the file name being taken from the original search term. Each thumbnail selected is given a name based on its category (rather than the original filename which is arbitrary and unlikely to be useful).

Figures 6 and 7 show the sequence of events for downloading thumbnail images. In Fig. 6 a search has been made for "British prime ministers", which returns Churchill, Tony Blair, Chamberlain and Robert Peel. An image has been selected from each row, and two (the first and fourth from left) from the Churchill row. (The last row is not visible in this screen-shot.) The "save" button has been clicked, bringing up a window confirming that five images will be saved in an archive file named "british_prime_ministers.zip".

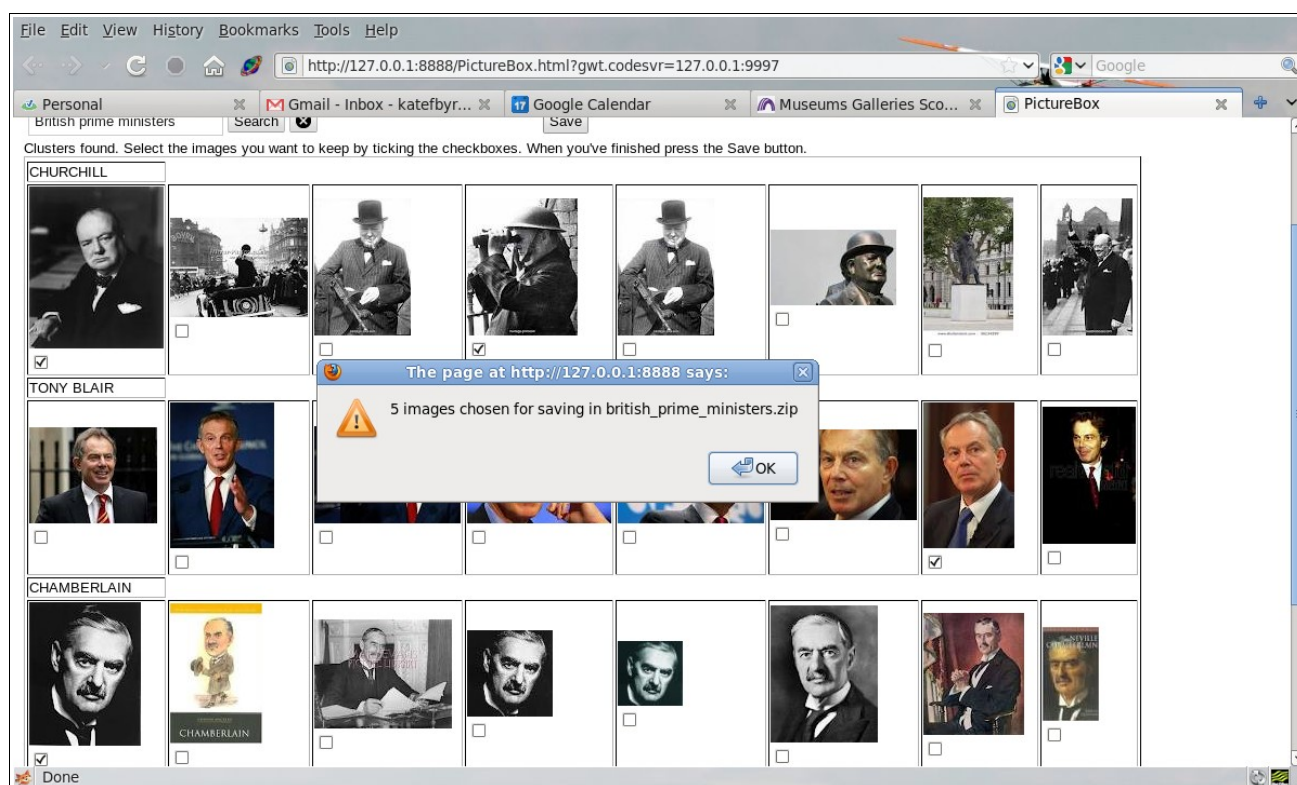


Figure 6: Five thumbnails selected from "British prime ministers" results.

When the alert window is dismissed by clicking "ok", the zipped archive is downloaded to the browser which handles it according to its configuration. In Fig. 7 the browser (Firefox) recognises the file type and offers to open the archive, with saving to disk as an alternative. This is typical browser behaviour, not generally requiring any configuration on the part of the user. As the pop-up window shows, at this point the browser is communicating with the server, which must obviously be configured for the purpose.

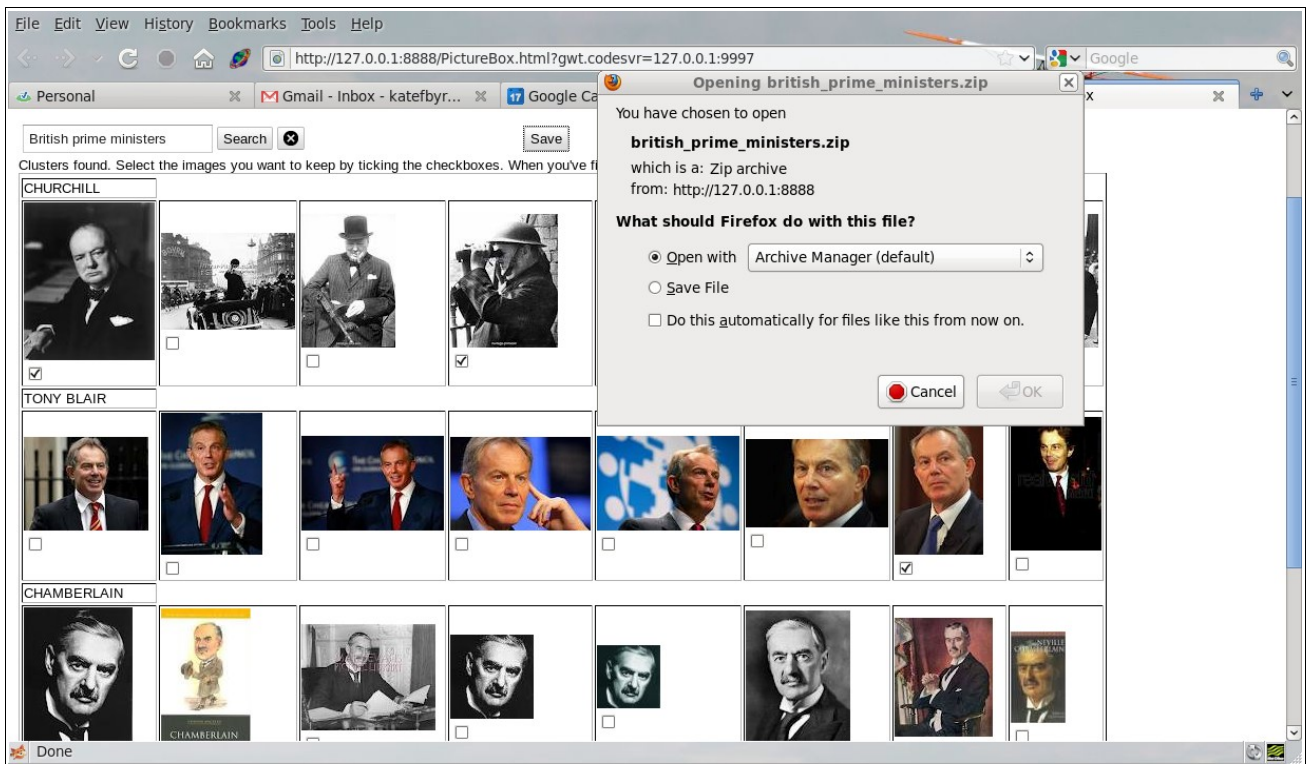


Figure 7: Saving or opening the zipped archive file.

5 Copyright and IPR

It is difficult to discuss images sourced from the Web without mentioning the thorny subject of copyright and intellectual property rights. When the user selects a number of thumbnail images for downloading, that's exactly what they get: thumbnails. Copying the thumbnails that are displayed in Google images and using them for educational purposes counts, I believe, as "fair use" (but I am not a lawyer). Following the links to find larger images – which will sometimes be available and sometimes not, and will often have copyright clauses attached somewhere on the page – would introduce a level of complexity that there was simply not enough time to attempt to deal with. For one thing one would have to look at resizing them for convenient use on screen, and there might also be a question about file sizes and data transfer rates. However, the main reason for not going beyond the thumbnail was that it seemed clear that one would risk copyright infringements and dealing properly with that possibility is a project in its own right.

The fact that most internet users probably infringe copyright in this way every day did not make me keener to write an application to help them! It seems quite probable that the current, almost unworkable, digital copyright regulations will change in due course, but at the moment this area is still a potential minefield for software developers. One possibly attractive option is discussed in Section 6 below: joining forces with Scran, who offer fully copyright-cleared resources to schools in Scotland.

6 Possibilities for Future Work

As mentioned above, the results are not comprehensive, nor would this be desirable. There is certainly scope for improvement but sadly shortage of time limited what could be achieved. When the collection of term-lists is collated into a single one some basic deduplication is done, as the most common terms are likely to appear on several of the lists. The deduplication needs improving – as it stands it fails to distinguish "eagle", "golden eagle" and "the bald eagle" for example. In this case it might make sense to attempt to recognise the most generic term, "eagle", and exclude it from the list. It would also be helpful to sort the list so that most frequently

occurring clusters were presented first; this proved surprisingly tricky to program efficiently and had to be skipped.

On the face of it, using Google, through the GWT functionality, to collect results should mean that one has so many available that spurious results could be weeded out reliably. One could, for instance, set a threshold and only keep terms for the final list that had appeared on at least two interim term-lists. However, the current GWT search functions severely limit the number of results returned so that this hoped-for feature was not available. This is quite a severe limiting factor and getting around it would unfortunately involve significant coding effort. Nevertheless, since the main objective of the project was to devise a method for returning clusters, and this has been done, it would not be impossible for a future release of the application to use the same programming logic (as shown in Fig. 4) but with a revised architecture.

It would probably be helpful if there were at least an option to download full size images instead of just thumbnails. As explained in Section 5, this was not implemented, mainly to avoid problems with copyright. The extra coding involved would not be very great, as the thumbnails come as a bundle of related links, including a pointer to the source page. There is an existing source of copyright-cleared graphical material that is widely used in schools, at least in Scotland, and that is Scran (<http://www.scran.ac.uk/>). It might be interesting to attempt to graft the idea behind *PictureBox* onto the Scran search interface. One would need to use the wider Google search for the first stage, to assemble the list of sub-category search terms, and these could then be applied against material where use of the images is fully licensed.

Finally, it would be very interesting to build a data set of accumulated search results, pairing terms with the narrower terms suggested by the system. This data set could be used to “grow” a knowledge base, that would have many of the characteristics of an ontology though the relationships between the broader and narrower terms would not always be straightforward “isa” instance relations (such as “apple is a fruit”). A “crowd-sourced ontology”, based around terms that users want to find instance clusters for, would be an interesting research resource. It would be a fairly trivial step to convert the accumulated data into an RDF graph, with nodes referring to existing image URLs. One can envisage using such a graph in a Linked Data setting, as a central reference or authority graph to connect distributed data. In discussion with Interface3 we have agreed that if the opportunity for such a project arises (ie there is any possibility of funding being available) they would be happy to see the project idea developed in this way.

Grateful thanks are due to EDTC Technology Gateway for providing the funding that made this project happen.

Kate Byrne

University of Edinburgh, October 2011.

References

- Harith Alani, Sanghee Kim, David E. Millard, Mark J. Weal, Wendy Hall, Paul H. Lewis, and Nigel R. Shadbolt. Automatic ontology-based knowledge extraction from Web documents. *IEEE Intelligent Systems*, 18(1):14–21, Jan/Feb 2003. URL <http://www.computer.org/intelligent>. doi:<http://doi.ieeecomputersociety.org/10.1109/MIS.2003.1179189>.
- B. Alex and C. Grover: Labelling and Spatio-Temporal Grounding of News Events, *Proceedings of the workshop on Computational Linguistics in a World of Social Media* at NAACL 2010, 6 June, 2010, Los Angeles, USA. URL <http://www.aclweb.org/anthology-new/W/W10/W10-0514.pdf>
- Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. DBpedia: A Nucleus for a Web of Open Data. *Proceedings of the 6th International Semantic Web Conference*, 4825:722–735, 2007. URL http://dx.doi.org/10.1007/978-3-540-76298-0_52.

- Christian Blaschke and Alfonso Valencia. Automatic ontology construction from the literature. *Genome Informatics*, (13):201–213, 2002.
- Douglas B. Lenat. CYC: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11):32–38, November 1995. URL <http://web.media.mit.edu/~lieber/Teaching/Common-Sense-Course/Lenat-CACM.pdf>
- George A. Miller. WordNet: a lexical database for English. *Communications of the ACM*, 38:39–41, November 1995. ISSN 0001-0782. URL <http://doi.acm.org/10.1145/219717.219748>.
- Ellen Riloff and Jeffrey Lorenzen. Extraction-based text categorization: Generating domain-specific role relationships automatically. In Tomek Strzalkowski, editor, *Natural Language Information Retrieval*, chapter 7, pages 167–196. Kluwer Academic, 1999. strz99.
- Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. YAGO: A Core of Semantic Knowledge Unifying WordNet and Wikipedia. In *Proceedings of the 16th international conference on World Wide Web (WWW2007)*, pages 697–706, Banff, Alberta, Canada, 2007. ACM. URL <http://www2007.org/papers/paper391.pdf>. ISBN: 978-1-59593-654-7.
- Viet-Ha Thuc, Jean-Michel Renders, Large-Scale Hierarchical Text Classification without Labelled Data, *Proceedings of the ACM Fourth International Conference on Web Search and Data Mining (WSDM 2011)*, February 9-12, 2011, Hong Kong. URL http://www.cs.uiowa.edu/~hviet/papers/wsdm_2011_Ha-Thuc.pdf