SubSift web services and workflows for profiling and comparing scientists and their published works

Simon Price[a,b,*], Peter A. Flach[b], Sebastian Spiegler[b], Christopher Bailey[a], Nikki Rogers[a]

[a] *Institute for Learning and Research Technology, University of Bristol, 8-10 Berkeley Square, Bristol BS8 1HH, UK*
[b] *Intelligent Systems Laboratory, University of Bristol, Merchant Venturers Building, Woodland Road, Bristol, BS8 1UB, UK*

Abstract

Scientific researchers, laboratories, organisations and research communities can be profiled and compared by analysing their published works, including documents ranging from academic papers to web sites, blog posts and Twitter feeds. This paper describes how the vector space model from information retrieval, more normally associated with full text search, has been employed in the open source SubSift software to support workflows to profile and compare such collections of documents. SubSift was originally designed to match submitted conference or journal papers to potential peer reviewers based on the similarity between the paper's abstract and the reviewer's publications as found in online bibliographic databases such as Google Scholar. The software is implemented as a family of RESTful web services that, composed into a re-usable workflow, have already been used to support several major data mining conferences. Alternative workflows and service compositions are now enabling other interesting applications, such as expert finding for the press and media, organisational profiling, and suggesting potential interdisciplinary research partners. This work is a useful generalisation and proof-of-concept realisation of an engineering solution to enable RESTful services to be assembled in workflows to analyse general content in a way that is not immediately available elsewhere. The challenges and lessons learned in the implementation and use of SubSift are discussed.

Keywords: Workflows, Web services, Ranking

1. Introduction

SubSift is a family of RESTful web services for profiling and matching text. Such services have numerous applications within e-Science and e-Research, both in terms of supporting administration of the scientific process itself and as a flexible processing component within data mining, machine learning and pattern analysis research. SubSift's URI design and built-in RDF support automatically connect its shared data to the Linked Data graph and wider Semantic Web.

In this paper we describe, through a variety of reusable workflows, how SubSift has been used to profile and compare scientists and their published works. These workflows were created as part of this work but the applications enabled by these workflows are not individually novel. However, collectively they demonstrate SubSift to be a useful generalisation and proof-of-concept realisation of an engineering solution to enable RESTful services to be assembled in workflows to analyse general content. To the best of our knowledge, despite the potential utility of such a set of services and notwithstanding that the underlying techniques are well established, such a solution is not immediately available elsewhere. These services are described and discussed in this paper at a level of detail not previously published elsewhere. Details of recently added services, such as polling and report publishing, or of SubSift's own workflow engine, have not previously been published.

The particular applications of SubSift described in this paper exploit the intuitive idea that the published works of scientific researchers, laboratories and organisations, in some sense, describe their specific research interests and expertise. By analysing these published works in relation to the body as a whole, discriminating profiles may be produced that effectively characterise heterogeneous documents ranging from traditional aca-

*Corresponding author. Tel.: +44 117 3314310; Fax: +44 117 3314396.

Email addresses: simon.price@bristol.ac.uk (Simon Price), peter.flach@bristol.ac.uk (Peter A. Flach), sebastian.spiegler@bristol.ac.uk (Sebastian Spiegler), c.bailey@bristol.ac.uk (Christopher Bailey), nikki.rogers@bristol.ac.uk (Nikki Rogers)

demic papers to web sites, blog posts and Twitter feeds. Such profiles have applications in their own right but can also be used to compare one body of documents to another, ranking arbitrary combinations of documents and, by proxy, individuals or groups by their similarity to each other.

The present SubSift was developed by generalising an earlier application-specific set of tools. Below we briefly review the original SubSift and its subsequent transformation into a prototypical workflow and a family of web services.

SubSift, short for *submission sifting*, was originally designed to match submitted conference or journal papers to potential peer reviewers based on the similarity between the paper's abstract and the reviewer's publications as found in online bibliographic databases. In this setting the software has already been used to support several major data mining conferences. The origin of the SubSift tools to support the research paper review process of the *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining 2009* (SIGKDD'09) and its subsequent evolution into RESTful web services[1] is documented elsewhere [1, 2]. In this paper we present this submission sifting use case as just one amongst a number of other re-usable workflows, where we define a *workflow* as follows.

Definition 1. Workflow *is a directed graph where vertices represent activities and edges represent data flow between activities. Activities (also referred to as components) are units of execution and may themselves be workflows. All workflows have one or more input ports and one or more output ports. Workflow enactment is the interpretation of the workflow graph by software (known as a workflow engine), executing activities in the required sequence to produce the described data flow.*

SubSift workflows are conceptual-level, graphical representations of workflow. They are visually inspired by Taverna workflow diagrams, but are not specific to any particular workflow enactment system. More formally, we define a *SubSift Workflow* as follows.

Definition 2. SubSift Workflow *is a schematic representation of a workflow where rectangular vertices represent activities enacted by SubSift web services, double rectangular vertices represent activities enacted asynchronously by SubSift background tasks, rounded vertices represent presentation-layer data transformation activities, and edges are labelled with (informal) data types. Activities enacted in SubSift services correspond to one or more SubSift REST API method calls. Input ports are represented as downward pointing triangles and output ports as upward pointing triangles. Data is delivered to input ports and received from output ports via HTTP.*

SubSift workflows have all been derived from the submission sifting design pattern shown in Figure 1. SubSift workflows in this paper extend this pattern at the input, e.g. harvesting documents from the web, or at the output, e.g. transforming similarity data into reports and graphs.

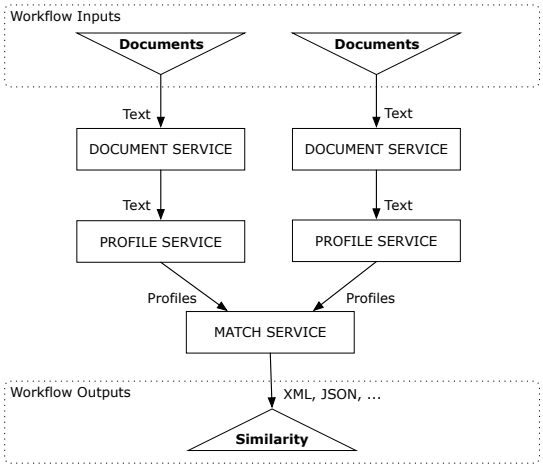

Figure 1: Design pattern schematic of original SubSift workflow.

SubSift is a freely available online application hosted by the University of Bristol. The software has also been released as open source on Google Code. Further details can be found on the SubSift website[2] along with extensive documentation and a growing number of demonstrations.

The remainder of this paper begins with a review of related work, followed by a recap of relevant background topics, an overview of the SubSift REST API, a detailed look at a number of re-usable workflows, a critical review and evaluation, before rounding up with future work and conclusions.

2. Related Work

As far as we are aware, the submission sifting use case for SubSift is a novel application of document sim-

[1]Funded by the UK's Joint Information Services Council (JISC) under the Rapid Innovation strand of the Information Environment Programme 2009-11

[2]`http://subsift.ilrt.bris.ac.uk`

ilarity methods. By contrast, the other use cases covered by the workflows in this paper have been the subject of prior work. Clustering data using similarity and distances is an active research topic in information retrieval and data mining [3, 4, 5] and has been applied before in e-Science and e-Research – for instance, a recent system automatically clusters search results according to the similarity of document content [6]. However, SubSift exposes its functionality through web services rather than locking it within a single application or experimental framework. The detailed discussion and demonstration of the flexibility of this approach is the main focus of this particular paper.

SubSift's functionality is achieved through the application of techniques from information retrieval that are more normally associated with *full text search* rather than profiling or matching entire collections of documents [7]. Established full text search tools such as Apache Lucene[3], recent versions of PostgreSQL[4] and Oracle UltraSearch[5] all support text matching on large-scale document collections. Although these systems were all designed to compare a single text query against a document collection, there is no reason why a full Cartesian product of one document collection against another document collection cannot be performed to return pairwise document similarity in the same way that SubSift does. Indeed, recently a paper duplicate detection system using Lucene [8] has been developed to do just that. However, implementing SubSift as a bespoke application has a number of advantages over full text search tools, including the following.

- Detailed metadata about factors contributing to term importance and similarity are available throughout.
- A consistent API is available for both the processing and the supporting data management, reporting and publication functionality.
- The core algorithm can be more easily modified or replaced without having to maintain patches or plug-ins for third-party tools.
- Control over the core algorithm and storage model is important for migration to an HPC setting.

Numerous text processing web services exist for specific tasks within a workflow, particularly domain-specific ones. Examples include the long-standing TerMine service for the recognition of multi-word terms, AcroMine for the expansion of bioinformatic abbreviations, and numerous others listed in the BioCatalogue life sciences web services directory [9, 10, 11]. However, SubSift services provide a service-based framework for managing and comparing collections of documents and for managing and publishing the results of analyses. To achieve the same with standalone text processing services would require the implementation of considerably more complex workflows. Text mining workflows incorporating web services can be constructed through U-Compare's ability to embed Unstructured Information Management Architecture (UIMA) components within Taverna workflows, but this is not itself a web service [12].

Recently, an information retrieval approach has been applied to web service discovery where the same vector space model as is used in SubSift has been shown to be effective as one of the components in ranking the similarity of web service descriptions [13]. Earlier work used similar techniques to construct semantic profiles of academics as the basis of a paper recommender system [14]. These systems differ from SubSift in that their functionality is neither general-purpose nor exposed as web services. Conversely, the SubSift-based *Submission Sifting* and the *Finding an Expert* workflows described later could be used to supply the text matching component of a web service discovery application or paper recommendation system respectively.

SubSift's RESTful services and the workflows presented in this paper are compatible with workflow management systems such as Kepler, Taverna and YAWL [15, 16, 17]. Compatibility would be further increased by the creation of a WSDL 2.0 description of SubSift's REST API which would add SOAP support to simplify integration into existing workflow design and enactment tools [18, 19].

3. Background

The theoretical basis for profiling and matching functionality of SubSift is the well known *vector space model* from information retrieval [7]. SubSift software also draws on the *representational state transfer* (REST) design pattern for web services [20]. In this section we give a brief overview of each of these topics. Readers already familiar with them can safely skip to the next section.

[3] `http://lucene.apache.org`

[4] `http://www.postgresql.org`

[5] `http://www.oracle.com/technology/products/ultrasearch/`

3.1. Vector Space Model

The canonical task in information retrieval is, given a query in the form of a list of words (terms), rank a set of text documents D in order of their similarity to the query. The vector space model is a common approach to solving this problem. Each document $d \in D$ is represented as the multiset of terms (bag-of-words) occurring in that document. The set of distinct terms in D, vocabulary V, defines a vector space with dimensionality $|V|$ and thus each document d is represented as a vector $\vec{d}$ in this space. The query q can also be represented as a vector $\vec{q}$ in this space, assuming it shares vocabulary V. The query and a document are considered similar if the angle θ between their vectors is small. The angle can be conveniently captured by its cosine, which is equal to the dot product of the vectors scaled to unit length, giving rise to the *cosine similarity*, s.

$$s(\vec{q}, \vec{d}) = \cos(\theta) = \frac{\vec{q} \cdot \vec{d}}{\|\vec{q}\| \cdot \|\vec{d}\|}$$

However, if raw term counts are used in vectors $\vec{q}$ and $\vec{d}$ then similarity will be biased in favour of long documents and will treat all terms as equally important. The *term frequency – inverse document frequency* (tf-idf) weighting scheme compensates for this by normalising term counts within a document by the total number of terms in that document, and by penalising terms which occur in many documents. More formally, *term frequency* tf_{ij} of term t_i in the document d_j, and *inverse document frequency* idf_i of term t_i are defined as

$$\mathrm{tf}_{ij} = \frac{n_{ij}}{\sum_k n_{kj}}, \quad \mathrm{idf}_i = \log_2\left(\frac{|D|}{\mathrm{df}_i}\right), \quad \text{tf-idf}_{ij} = \mathrm{tf}_{ij} \times \mathrm{idf}_j$$

where *term count* n_{ij} is the number of times term t_i occurs in the document d_j, and *document frequency* df_i of term t_i is the number of documents in D in which term t_i occurs.

In SubSift, instead of comparing a single query against a set of documents, we pairwise compare every document in one collection D_1 (e.g. abstracts) with every document in another collection D_2 (e.g. reviewer bibliographies) to produce a ranked list for each document. To capture the overall importance of each term across the combined collections, df_i, and hence tf-idf_i, values are calculated over the union of both collections, $D_1 \cup D_2$.

3.2. RESTful Web Services

REST is an easily understood design pattern for web services, based around the ubiquitous HTTP protocol and its familiar vocabulary of URIs, media types, requests and responses [20]. Recently, REST web services have become popular as the web API behind numerous Web 2.0 sites, including Twitter, Flickr and Facebook and are also widely used in e-Science and e-Research. Like conventional websites, *RESTful* services offer a stateless, cacheable, layered and uniform client-server interface. However, unlike conventional sites, which are designed to render human-readable data as HTML pages to be viewed in a browser, RESTful sites serve data in formats such as XML, JSON and RDF, that may be readily consumed by arbitrary applications. Furthermore, in the same way that HTML forms on conventional web pages can be used to submit data from the client to the server for storage and processing, arbitrary RESTful applications can use exactly the same protocols to achieve the same end. Also, the usual HTTP authentication and authorisation mechanisms can be used to control access to specific services and resources.

The intuition behind REST is that URIs are used to represent resources and that HTTP request methods are used to specify fundamental operations on those resources. The most widely used HTTP request method, `GET`, is invoked every time a web browser requests a URI from a web server. HTTP `GET` is only one of several HTTP request methods; the five that are most significant for REST web services are summarised in Table 1. These correspond to the *CRUD(E)* (Create, Replace, Update, Delete and Exists) operations from Web 2.0. Additional operations are specified by adding verbs into the URIs themselves. Pairs of attribute-value parameters can be supplied with the HTTP request to modify the behaviour of operations.

The following is an example of a REST request, using `curl`[6] as a convenient command line client, to retrieve a (usually XML) representation of a book by its ISBN.

```
curl -X GET http://www.foo.com/books/0471941522
```

A similar request to the same URI using the `DELETE` method will delete that same representation.

```
curl -X DELETE http://www.foo.com/books/0471941522
```

As the second example makes clear, authentication and authorisation are sometimes required for REST operations. All the usual HTTP techniques can be used for this purpose.

[6] `http://curl.haxx.se`

Table 1: Interpretation of HTTP Request Methods in REST

Method	Usage in REST	Read-only
`GET`	*show* and *list*	yes
`HEAD`	*exists* (resource exists?)	yes
`POST`	*create* and *compute*	no
`PUT`	*update* and *recompute*	no
`DELETE`	*destroy*	no

4. SubSift Web Services

The decision to package and host the SubSift software as web services was easily made. Apart from the advantage of prospective users being able to use the software without having to install it, the web services model has a number of other benefits, including the ability for users to: make their own application-specific customisations using the services with mashup tools like Yahoo Pipes; select other data sources, such as Citeseer, Google Scholar, eprints, news and blogs; and, importantly, integrate the software with their own research tools without having to modify code.

The choice of RESTful web services over SOAP-based web services was more difficult. Popular e-Science workflow tools, such as myExperiment [21] and Taverna [16] support both REST and SOAP to varying degrees. However, SOAP is well established in e-Science, as evidenced by the BioCatalogue[7] life sciences web services directory which, at the time of writing, has 1974 SOAP services as compared to only 72 REST services. Even so, the compelling argument for implementing SubSift as lightweight RESTful services was our strong use cases for Web 2.0 applications and mashups. For developers in general there is a growing trend away from SOAP services towards the simpler RESTful ones, for all the reasons espoused in [22], and this was also an important factor in our choice.

Figure 2 shows the high-level design of the SubSift system, with its REST API and supporting web harvester robot. Details of the API itself are given in the next section and further details appear in the workflow descriptions below.

5. SubSift REST API

The SubSift REST API is organised around a series of *folders* into which data *items* are stored. This organisation is modelled on the familiar filing system concept

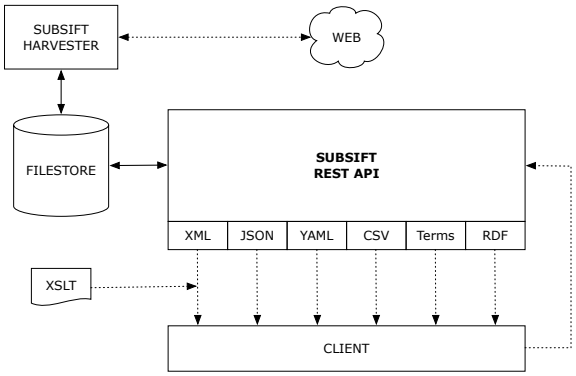

Figure 2: SubSift System Architecture. Dotted lines represent HTTP. Solid lines represent file access. XML, JSON, YAML, CSV, Terms and RDF are response formats. Terms are Prolog terms. XSLT stylesheets are invoked via optional inclusion of XSL processing command in XML returned by SubSift.

of folders and files. The three main folder types are documents, profiles and matches. The first request below would create a documents folder called `staff` and the second would list its items[8].

```
curl -X POST <uri>/<user_id>/documents/staff
curl -X GET <uri>/<user_id>/documents/staff/items
```

In SubSift, a document is a piece of text to be profiled and matched. A document will usually be the text from some external source such as the text of a web page or a conference paper abstract. A profile is a summary representation of the features of a single document, with respect to the other documents in the same documents folder. In the example fragment of a profile item below, the person's names are the most discriminating terms, occurring 77 times in the source DBLP author webpage[9] and having a tf-idf of 0.404 (last name) and 0.246 (first name) relative to the rest of the items in this profiles folder. The terms *logic* and *ILP* also rank highly.

```
profile:
  name: Peter Flach
  source: http://dblp.../Flach:Peter_A=.html
  term:
    - name: flach
      idf: 6.047
      n: 77
      tf: 0.067
```

[7] BioCatalogue website – http://www.biocatalogue.org/services

[8] We omit the security token needed in the request header of `DELETE`, `POST` and `PUT` requests, and of all requests for folders marked as private.

[9] `http://dblp.uni-trier.de/`

```
    tfidf: 0.404
  - name: peter
    idf: 3.685
    n: 77
    tf: 0.067
    tfidf: 0.246
  - name: logic
    idf: 3.924
    n: 18
    tf: 0.016
    tfidf: 0.061
  - name: ilp
    idf: 4.685
    n: 14
    tf: 0.012
    tfidf: 0.057
```

One usage of profiles in SubSift is to obtain a list of distinguishing terms, or keywords, for a document – for example, automatically extracting keywords from abstracts of papers submitted to a conference. Another usage is for two profile folders to be compared against each other to produce a matches folder. A matches folder is created by analysing every pairing of profile items drawn from the two profiles folders. Each match item records the tf-idf cosine similarity, and various related statistics, of a single profile from the first profiles folder against every profile from the second profiles folder. A typical usage of such a comparison is to match submitted conference paper abstracts with the bibliography pages of Programme Committee (PC) members (i.e. reviewers) in order to rank potential reviewers for each paper and vice versa, as depicted in Figure 3.

SubSift's API methods make intermediate data and metadata available for all folders and items. For example, the entire similarity matrix of a match folder can be exported or the relative contribution of each term towards a particular match item's similarity can be retrieved. In the match item fragment below, metadata about the similarity score of 0.076, between a person and a paper, shows that the terms *logic* and *ILP* made large contributions to the score relative to other terms.

```
match:
  name: Peter Flach
  source: http://dblp.../Flach:Peter_A=.html
  item:
    - description: paper title removed
      name: paper id removed
      score: 0.076
      source: text
      term:
        - name: logic
          contribution: 0.005
        - name: ilp
          contribution: 0.004
        - name: class
          contribution: 0.001
```

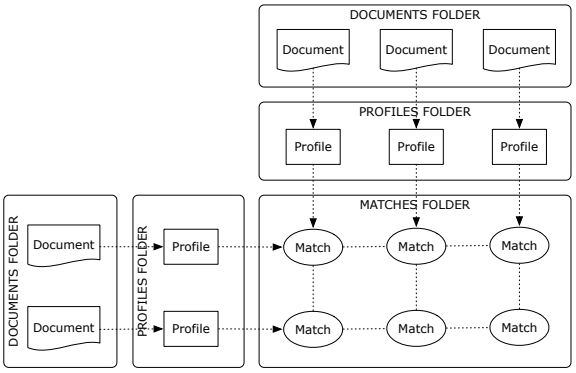

Figure 3: A SubSift REST API controlled sequence of transformations from a pair of document folders (e.g. abstracts versus reviewers) through to a folder of matching statistics. Original and computed data plus metadata can be obtained at each step in the process via API methods.

```
  - name: inductive
    contribution: 0.001
```

For ease of integration, there is flexibility in both input and output. Document text may be added per item or in bulk or by supplying a list of URLs to be fetched asynchronously by SubSift's harvester robot. The API methods can return data in the following representational formats: CSV, JSON, RDF, XML, YAML, and Prolog terms.

Below we give a flavour of how the workflows in this paper can be enacted through sequences of REST method calls to SubSift's web services. As our example we choose the canonical *Submission Sifting* workflow. In steps 1–5, profiles of conference PC members are created; in steps 6–8, profiles of the submitted abstracts are created; and in steps 9–10, the two sets of profiles are pairwise matched against each other and the results published to the web. For readability, at each step the HTTP request methods and their parameters are denoted using the format below and no response values are shown.

```
<http_method> [<uri>][<user_id>]<path>
<parameter_name_1> = <parameter_value_1>
<parameter_name_2> = <parameter_value_2>
...
<parameter_name_N> = <parameter_value_N>
```

For brevity we will omit `<uri>`, which has the value `https://subsift.ilrt.bris.ac.uk` for the publicly hosted version of SubSift, and `<user_id>` which will always be the account name (e.g. `kdd09` for the SIGKDD'09 conference). Note also that we omit details of the security token needed in the HTTP request

header of all DELETE, POST and PUT requests. The token is also required to access folders and data marked as private, irrespective of request method.

Step 1. Obtain a list of PC member names and their DBLP author page URIs. SubSift's DBLP Author Finder demo accepts a list of author names and then looks up these names on the DBLP Computer Science Bibliography and suggests author pages which, after disambiguation, are returned as a list with each line as: <pc member name>, <uri>. Create a bookmarks folder to hold the list of PC member URIs.

```
POST /bookmarks/pc
```

Step 2. Create bookmarks in this folder, one per PC member URI.

```
POST /bookmarks/pc/items
items_list=<list of URIs from step 1>
```

Step 3. Create a documents folder to hold the web page content (text) of the DBLP author pages.

```
POST /documents/pc
```

Step 4. Import the bookmarks folder into the documents folder. This adds the URIs to SubSift Harvester Robot's crawl queue. We name the documents folder the same as the bookmarks folder. This is a convention, not a requirement, but makes the ancestry of the folder obvious.

```
POST /documents/pc/import/pc
```

In time, all the URIs will be fetched and a document created in the documents folder for each webpage fetched. To detect if there are still URIs waiting to be fetched, applications may poll the same URI until an HTTP 404 error is returned.

```
HEAD /documents/pc/import/pc
```

Step 5. Create a profiles folder from the bookmarks folder.

```
POST /profiles/pc/from/pc
```

Step 6. For bulk upload, pre-process the abstracts into CSV format so that each line is: <paper id>, <abstract>. Include the text of the paper title in with the abstract text. Create a documents folder to hold the abstracts.

```
POST /documents/abstracts
```

Step 7. Use the abstracts CSV text to create a document item for each abstract.

```
POST /documents/abstracts/items
items_list=<csv from Step 6>
```

Step 8. Create a profiles folder from the documents folder.

```
POST /profiles/abstracts/from/abstracts
```

Step 9. Match the PC members profiles folder against the abstracts profiles folder.

```
POST /matches/pc_abstracts/profiles/pc/with/abstracts
```

Having generated the matches data it may then be retrieved in a variety of ways. For example, to fetch the ranked list of papers per PC member and then to fetch the ranked list of reviewers per paper and finally retrieve the similarity matrix to use for bidding, optionally specifying manually chosen thresholds to discretize the scores into the range 3..1 as bid values.

```
GET /matches/pc_abstracts/items
profiles_id=pc

GET /matches/pc_abstracts/items
profiles_id=abstracts

GET /matches/pc_abstracts/matrix
```

Step 10. Generate and publish profiles and matches as human-readable reports to the web or download them as a zip of webpages to publish on the user's own site.

```
POST /reports/pc/profiles/pc
POST /reports/abstracts/profiles/abstracts
POST /reports/pc_abstracts/matches/pc_abstracts
```

The published reports appear on the web at /reports/pc, /reports/abstract and /reports/pc_abstracts and are either privately or publicly accessible.

6. Demonstration Workflows

In the following sections we present a number of SubSift workflows as use cases illustrated by data flow diagrams. This presentation avoids details of specific workflow runtime environments and at this level of abstraction, the workflows are highly re-usable.

6.1. *Workflow 1 – Submission Sifting*

Peer review of written works is an essential pillar of the academic process, providing the central quality control and feedback mechanism for submissions to conferences, journals and funding bodies across a wide range of disciplines. However, from the perspective of a busy conference PC chair, journal editor or funding manager, identifying the most appropriate reviewer for a given submission is a non-trivial and time-consuming task. Effective assignment, first and foremost, requires a good match to be made between the subject of the submission and the corresponding expertise of reviewers drawn from a, sometimes large, pool of potential reviewers. In the case of conferences, a recent trend transfers much of this allocation work to the reviewers themselves, giving them access to the full range of submissions and asking them to bid on submissions they would like to review. Their bids are then used to inform the allocation decisions of the Programme Committee chair.

A SubSift demonstrator based on this workflow was implemented as a wizard-like series of web forms, taking the PC chair through the above process form by form. On the first form, a list of PC member names is entered. SubSift looks up these names on DBLP and suggests author pages which, after any required disambiguation, are used as documents to profile the PC members. Then the conference paper abstracts are uploaded as a CSV file and their text is used to profile the papers. After matching PC member profiles against paper profiles, SubSift produces reports with ranked lists of papers per reviewer, and ranked lists of reviewers per paper. Optionally, by manually specifying threshold similarity scores or by specifying absolute quantities, a CSV file can be downloaded with initial bid assignments for upload into the conference management tool.

The data flow details for this workflow are depicted in Figure 4. This is almost the generic workflow from Figure 1, apart from the inclusion of the Bookmarks Service to manage the list of URIs and SubSift's Harvester Robot to fetch the web pages. The HTML Generators use XSLT to transform XML into HTML reports.

This workflow was first used at SIGKDD'09 and quantitatively evaluated by comparing SubSift's initial bids against the revised final bids made by reviewers [1]. The results plus qualitative feedback from reviewers, were encouraging and demonstrated that there is considerable scope for automated support during the bidding process.

More recently, a Web 2.0 implementation of this workflow has been released on the SubSift website as a publicly available demonstration and as a practical tool

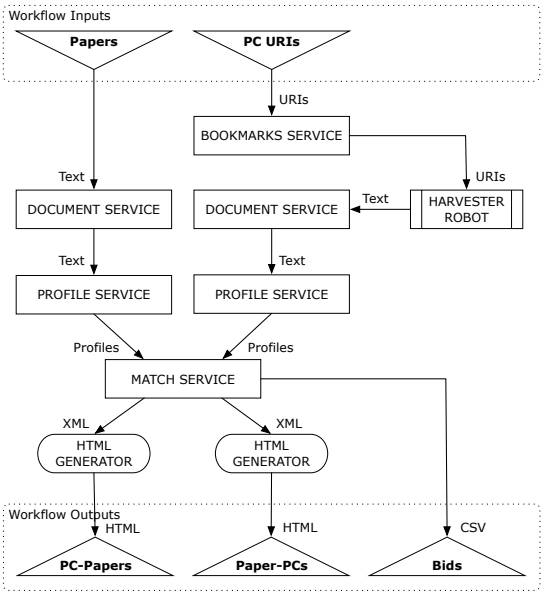

Figure 4: Schematic of *Submission Sifting* workflow.

for conference organisers, in this example, from the domain of computer science. The earlier SubSift demonstrator did not make use of web services and was instead a single self-contained web application. The new Submission Sifting demonstration is an entirely client-side JavaScript workflow enactment which integrates calls to the SubSift web services with calls to additional web services for disambiguating DBLP author homepages, extracting publication details from DBLP author pages and for aggregating multiple web pages into a single *bag of words*. An example of the wizard-style dialogue presenting the user with disambiguation choices is shown in Figure 5.

6.2. *Workflow 2 – Finding an Expert*

Finding an expert, within an organisation, on a given topic can be difficult if the person searching is not familiar with all members of that organisation or with the topic itself. This is often true for members of the media seeking an expert, within a university, on some newsworthy scientific report. It is also true for journal editors deciding which editorial board members to ask to review a particular paper.

In this workflow the user submits a fragment of text, typically an abstract or the full text of a paper, and SubSift compares this against the publications of a predefined list of researchers. The result is a list of researchers ranked by their similarity to the submitted fragment. Figure 6 shows *ILRT Matcher*, an entirely

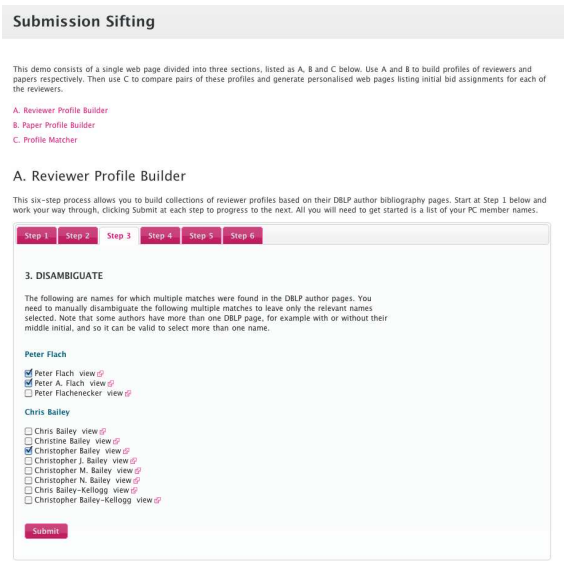

Figure 5: Web 2.0 wizard implementation of the *Submission Sifting* workflow, showing the Disambiguation step of the Reviewer Profile Builder. In the example, Peter Flach and Peter A. Flach are both selected because both of the corresponding DBLP pages refer to the same author.

client-side, Web 2.0 JavaScript implementation of this workflow, comparing the submitted text against every ILRT staff members' homepage. The results are displayed as a bar chart. Optionally, the list of terms contributing to each researcher's similarity score can be viewed along with the percentage that each term contributed to the combined score, as shown in Figure 7.

This two-phase workflow is depicted in Figure 8. The first phase is preparatory and creates all the necessary folders and items for the second, interactive phase. The second phase is enacted each time a user submits text

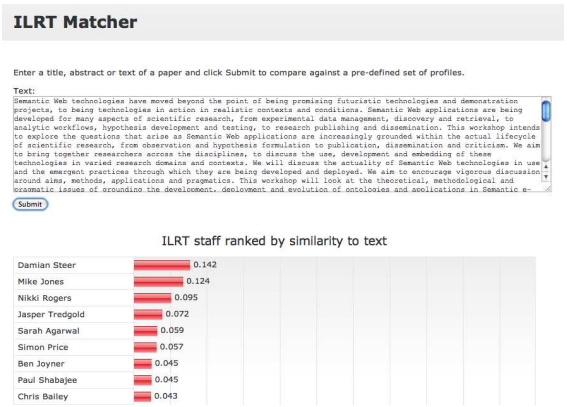

Figure 6: Similarity match between text fragment and ILRT staff.

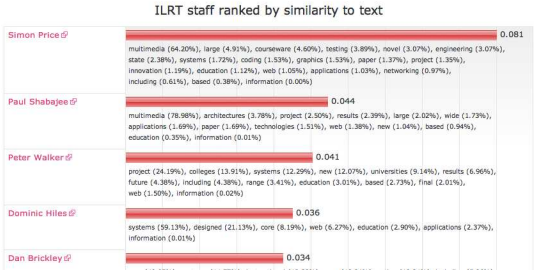

Figure 7: Term contributions for matches between text and ILRT staff.

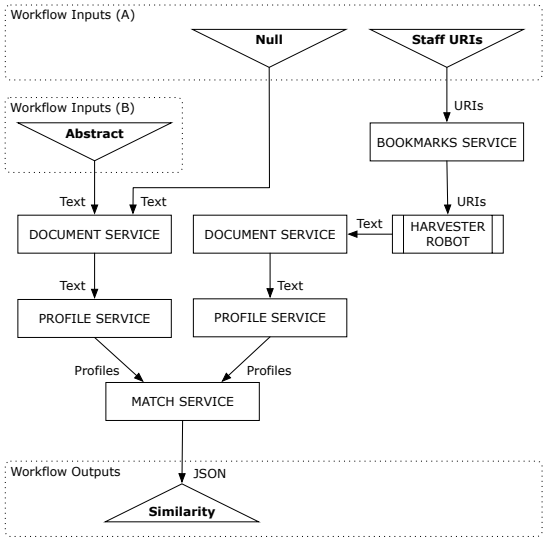

Figure 8: Schematic of *Finding an Expert* workflow.

via the Web 2.0 form. In the first phase a null (empty) text fragment is used to form a singleton item in a documents folder to be profiled and compared against the staff web page profiles.

6.3. Workflow 3 – Visualising Similarity

Organisations usually have a formal hierarchical structure but there can also be hidden structures, both of which can be discovered by analysing the similarity of documents associated with members of the organisation. The similarity between individuals can be represented as a directed labelled graph where the vertices are the individuals and the edges the similarity relation between two individuals. One strategy for making structure visible is to only add edges between nodes whose similarity score is above a threshold. Another is to only add the top n edges for each node, where n is a small integer. Tools like Graphviz[10] can be used to visu-

[10] `http://www.graphviz.org`

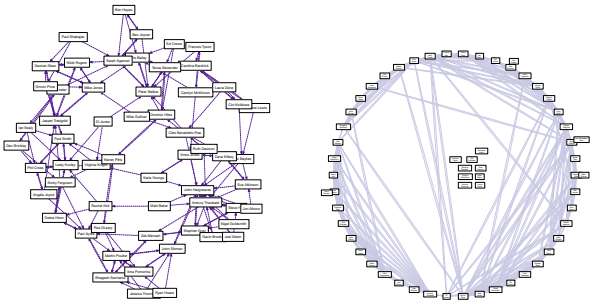

Figure 9: ILRT staff homepage similarity matches data output to DOT format and rendered in Graphviz (left: force-directed layout, right: circular).

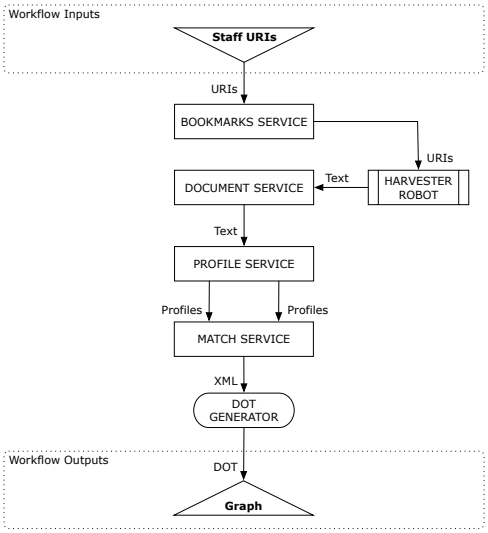

Figure 10: Schematic of *Visualising Similarity* workflow.

alise a textual representation (DOT format) of the graph as shown in Figure 9 for the homepages of ILRT staff members.

This workflow in Figure 10 compares a profiles folder against itself to rank all the documents profiled by their similarity to each other. Ignoring the reflexive matches (a profile is always identical to itself), the resultant similarity data returned by SubSift is then transformed by a SubSift XSLT file from XML into a textual DOT file suitable for rendering as an image in Graphviz.

6.4. *Workflow 4 – Profiling Reading Lists*

Many researchers share lists of bookmarks (URIs) to social bookmarking websites like Delicious[11]. As their

[11] `http://delicious.com`

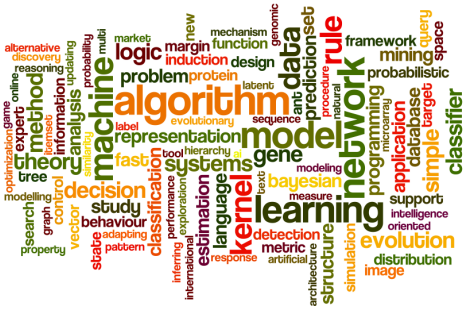

Figure 11: Tag cloud visualisation of researcher interests.

interests change over time, so does the list of bookmarks. The bookmarks and the text of the web pages that they link to can be used to create a profile of a researcher's current interests. Such information is often displayed on homepages as a tag cloud, such as Figure 11 which was produced in Wordle[12] using SubSift profile terms. This information may also be published as Semantic Web data using formats like Friend of a Friend (FOAF).

A SubSift workflow to achieve this is shown in Figure 12. The bookmarks are harvested to create the researcher's web reading list profile. The researcher's bookmarks are not being compared to those of other researchers here and so the ranked list of terms produced will contain high scoring terms, such as the person's name, that are not relevant. To filter these out, the terms of the ACM Computing Classification System (CCS)[13] are used during profiling to ensure that all terms output from the Profile Service occur in the CCS list.

6.5. *Workflow 5 – Ranking News Stories*

A researcher's subscriptions to RSS blog and news feeds provide frequently updated textual information. However, reading RSS feeds can be time consuming and what may initially appear relevant from the title may turn out not to be after following the associated link to the full text. The ability to rank stories in a feed by relevance to the research interests of the reader is a potential time saver.

Figure 13 shows a two-phase workflow based on an extension to the submission sifting workflow that can be used to reorder the stories in an RSS file according to their similarity to the researcher as defined by their own published works. In the first phase, the current homepage of the researcher is used to create their profile. This profile only needs recalculating periodically.

[12] `http://wordle.net`

[13] `http://www.acm.org/about/class/1998/`

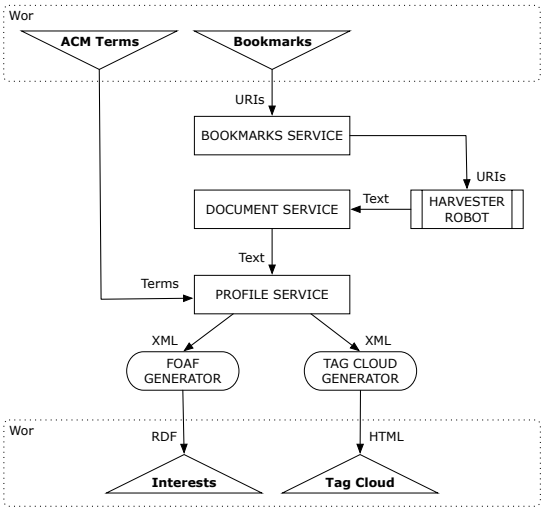

Figure 12: Schematic of *Profiling Reading Lists* workflow.

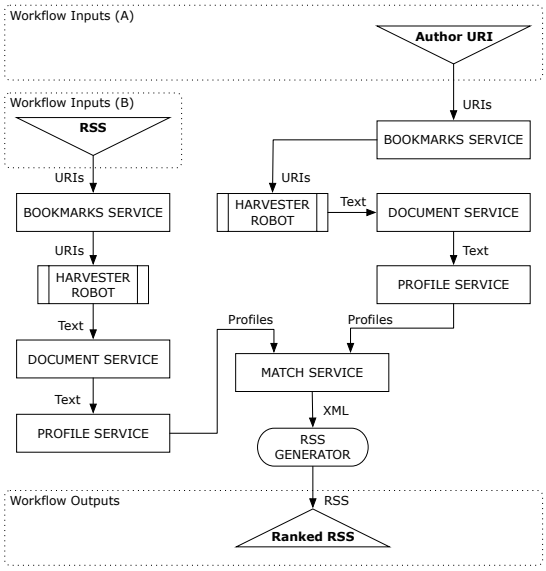

Figure 13: Schematic of *Ranking News Stories* workflow.

In the second phase, an RSS feed is used as a list of URIs which are harvested and used to create a profile for each story. The story profiles are matched against the reviewer profile and the resultant ranking is used to reorder the original RSS file.

7. Critical Review and Evaluation

The SubSift workflows described in the previous section demonstrate the flexibility and utility of SubSift's web services. In this section we reflect more widely on how SubSift services have been used in practice, their ease of use, the challenges and lessons learned through their use. We also discuss steps taken in hindsight to overcome some of the most serious challenges encountered in using these services.

7.1. Learning to Use SubSift

One aspect of SubSift that is notably successful in practice is its learnability. Positive feedback has been received from both research students and software professionals about, in particular, the documentation and the SubSift REST API Explorer demonstration[14]. The former extends to around 80 pages with examples and background information covering the basics of HTTP and REST. The latter enables API methods to be called interactively using a dynamic web form.

7.2. Using SubSift from Web 2.0

The primary use envisioned for SubSift's web services was as the server-side processing behind a browser-based Web 2.0 application, specifically designed to support academic peer review. This has now been realised and is available as an online demonstration[15]. The original submission sifting demonstrator was as a traditional forms-based website, with CGI scripts handling form submissions on the server. Reimplementing this as a Web 2.0 application using SubSift services added several previously missing features and, based on informal feedback, has substantially improved the user experience. However, leaving aside the additional functionality, simply reimplementing the old functionality through AJAX proved substantially more time consuming than anticipated due to a number of factors discussed below.

Dependencies between REST method calls require that one call finish before another is issued, even though AJAX calls are, by definition, asynchronous. A typical example is: the call to create a documents folder must complete before a call to import documents into that folder is issued. The current generation of JavaScript libraries do not support multiple-step AJAX operations and so, to keep within the time constraints of the project, we implemented each REST method call individually with its own asynchronous error and success handling.

[14] `http://subsift.ilrt.bris.ac.uk/demo/explorer`

[15] `http://subsift.ilrt.bris.ac.uk/demo/sift`

This does not make the underlying workflow being enacted via the JavaScript code particularly clear, in comparison to a conventional workflow language. One solution would be to invoke SubSift's services indirectly through a workflow engine, one with a web services API and also capable of invoking RESTful services. However, this same problem occurs in multiple general-purpose programming languages, not just JavaScript, and so we moved the problem from the client language to the server, by adding lightweight workflow enactment support to SubSift. This capability, described later, came too late for the current Web 2.0 submission sifting implementation but has now been used successfully from Web 2.0 for an in-house administration tool.

Cross-domain browser security policy makes it more difficult to issue the range of HTTP Methods used in REST method calls to the SubSift API domain from the different domain the Web 2.0 website was originally hosted on. Workarounds include proxying or same-domain hosting. We opted for the latter and moved the website to the same domain, but this is not a generally practical solution for Web 2.0 applications. Eventually, HTML 5 will make this problem much more manageable, but in the meanwhile server-side proxying or non-portable scripting is required for convenient cross-domain REST method calls from a browser.

File upload support was not initially available in SubSift services. Handling the conference paper abstracts csv file upload at the server, without resorting to separate bespoke server-side scripts, required new features to be added to SubSift. As a consequence, file upload support is now transparently supported as standard for all SubSift API parameters, across all API methods.

In amongst the above problems, one real design success has been SubSift's inbuilt support for JSON, which has made the handling of data returned by the web services trivial here and in the various Web 2.0 applications developed to-date.

7.3. Using SubSift from an HPC Research Tool

Three of the SubSift workflows (Submission Sifting, Finding an Expert, and Visualising Similarity) have now also been made available through the prototype Exabyte Research Hub[16] data mining and machine learning framework. This HPC tool presents a consistent and easy-to-learn web interface designed to facilitate the use and interaction of a growing set of analytical tools, of which SubSift is the latest addition (Figure 14).

[16] `http://exabyte.bris.ac.uk/hub.html`

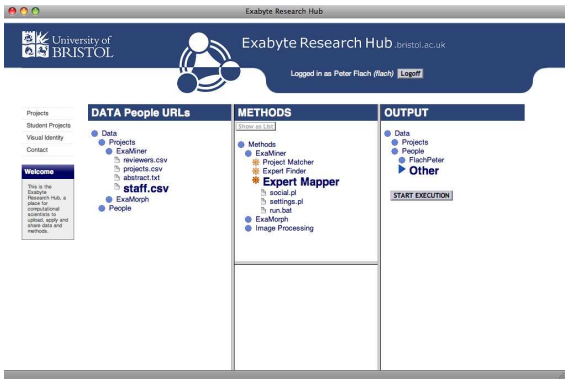

Figure 14: SubSift workflows deployed as flexible processing modules within a web client to the Exabyte Research Hub.

Each of these workflows was implemented as a command line executable program, complete with help text, parameter validation and error handling for failed web service calls. These programs, although created in Perl, encountered exactly the same REST method call dependency issues as were encountered in the Web 2.0 JavaScript setting. In hindsight, the now extant lightweight workflow enactment support in SubSift would have significantly reduced the amount of scripting required to enact the workflow.

7.4. Responding to Lessons Learned

The need to support multiple-step REST method calls is one of the most important lessons learned. Incorporating this capability into SubSift itself avoids introducing a dependency on external workflow engines, for even simple use cases such as those in this paper. To-date SubSift has been used from Java, JavaScript, Perl, Python and SWI-Prolog, which all require their own approaches to executing sequences of REST API calls. Avoiding the need to implement these in general-purpose languages has resulted in the more recent applications and tools requiring much less coding to implement. The following brief examples gives a flavour of SubSift's lightweight workflow language for encoding simple sequences of commands.

```
?, delete, documents/pc
+, post, documents/pc, mode=private
+, post, documents/pc/import/pc
*, head, documents/pc/import/pc
?, delete, profiles/pc
+, post, profiles/pc/from/pc, "ngrams=1,2"
?, delete, matches/pc_abs
+, post, matches/pc_abs/profiles/pc/with/abs
```

Each line corresponds to a single API method call. The above example deletes and creates/recreates a documents folder called `pc` into which it imports the bookmarks from bookmarks folder `pc`. It then profiles the `pc` documents folder. A match folder is similarly deleted and created/recreated before, finally, a match is performed between the `pc` and `abstracts` profile folders to create the `pc_abs` matches folder. Each command will wait for the previous command to complete - including waiting for all the bookmarks imported into the `pc` folder to be harvested by the web robot (which may take minutes or hours). The action taken in response to the HTTP status returned by each call is determined by the symbol at the start of the line. '`+`' requires an HTTP success code to continue, '`-`' requires a failure code, '`?`' ignores the code, and '`*`' repeats the same command until a failure code is returned. The '`*`' code enables polling to be performed using predicate methods recently added to SubSift for this precise purpose.

This lightweight workflow representation is executed by an equally lightweight workflow engine. However, the addition of this capability enables the above sequence of REST method calls to be made by a single REST call, greatly simplifying the use of SubSift from general-purpose languages and, for that matter, from specialist workflow languages. Other non-trivial REST APIs might also benefit from a similar approach.

8. Future Work

Use cases for existing SubSift workflows have emerged that require large-scale storage and High Performance Computing (HPC) capability – for instance, to profile and compare local document collections against all the documents in a sizeable online bibliography such as DBLP. As part of the University of Bristol's Exabyte Informatics research programme, parts of SubSift's functionality are being evaluated for migration to run on the University's BlueCrystal HPC facility [17] and Petascale storage.

However, the workflows presented in this paper demonstrate practical applications of SubSift on small data sets with modest hardware. We anticipate that these workflows will be used largely unchanged in the HPC setting; only the internal behaviour of SubSift will differ, most likely being re-implemented using the MapReduce [23] paradigm in Apache Hadoop[18]. The development of these workflows with the current small-scale implementation has elicited valuable use cases from researchers and research managers; it has also served to refine the API requirements for different runtime settings, such as invocation from Web 2.0 pages or interaction with tools such as Matlab. Even at this current scale, SubSift is being used to provide expert profiling, finding and publication matching as part of the ResearchRevealed[19] project which aims to provide customisable access to integrated institutional research repositories and views of the real impact of research in its wider context, for example in industry, in society, and across the UK research community.

Both small-scale and HPC branches of SubSift will contribute towards the development of ExaMiner, a prototype application for mining and mapping a university's research landscape, beginning initially with the University of Bristol but with the ultimate aim of being able to map any research-centric institution. Specifically, the ExaMiner software will: generate automatic profiles of researchers and research groups from university web pages; allow visual navigation of these profiles, as well as navigation by free-text query; provide social networking components, including the ability to store a personal profile and a network of interesting contacts, events and connections; notify a user of new resources in their sphere of interest; and improve performance over time by responding to user feedback.

Possible users include academics, research managers, current and prospective students, and the media. It is hoped that a system based on this prototype would improve the discoverability of research within the institution, the community, the media, prospective students, and the government.

9. Conclusion

SubSift's RESTful web services show potential for integration into workflows far beyond the original peer review use case where SubSift has already proved to be a popular addition to the existing conference paper reviewing process. While serving primarily to demonstrate the utility of SubSift, we suggest that, in their own right, the workflows presented in this paper are promising e-Science and e-Research use cases for profiling and comparing scientists and their published works. In this

[17] BlueCrystal features: 416 nodes each with 2.8 GHz Intel Harpertown E5462 processors, memory 8 GB RAM per node (1 GB per core); QLogic Infinipath high-speed network; IBM's General Parallel File System (GPFS) providing data access from all the nodes; 73TB SATA data storage.

[18] `http://hadoop.apache.org/`

[19] `http://researchrevealed.ilrt.bris.ac.uk/`

paper we have described a number of previously unpublished features of SubSift and provided the most in-depth discussion and analysis of the system to-date.

We also forward SubSift as a flexible tool for constructing, manipulating and publishing document-centric datasets for information retrieval, data mining and pattern analysis research. The publication of research data in this way supports the reproducibility of scientific experiments, while automatically connecting data through Linked Data and the Semantic Web.

Acknowledgment

We would like to thank conference PC chairs Bart Goethals (SDM'10), Qiang Yang (KDD'10), Mohammed Zaki (PAKDD'10) and Geoff Webb (ICDM'10) for trialing SubSift; other developers of the original software, Bruno Golénia from the Intelligent Systems Lab in Bristol, John Guiver, Ralf Herbrich and Thore Graepel from Microsoft Research in Cambridge, and Mohammed Zaki from Rensselaer Polytechnic Institute and Mike Jones, Damian Steer and Jasper Tredgold of ILRT for discussions about RDF support.

References

[1] Flach, P.A., Spiegler, S., Golénia, B., Price, S., Ralf, J.G., Graepel, H.T., et al. Novel tools to streamline the conference review process: Experiences from SIGKDD'09. SIGKDD Explorations 2009;11(2):63–67.

[2] Price, S., Flach, P.A., Spiegler, S.. SubSift: a novel application of the vector space model to support the academic peer review process. In: Workshop on Applications of Pattern Analysis (WAPA 2010). Windsor, UK; 2010,.

[3] Newman, M.E.J.. The structure of scientific collaboration networks. Proceedings of the National Academy of Sciences 2001;98(2):404–409.

[4] Gaertner, T., Lloyd, J.W., Flach, P.A.. Kernels and distances for structured data. Machine Learning 2004;57(3):205–232.

[5] Shawe-Taylor, J., Cristianini, N.. Kernel Methods for Pattern Analysis. Cambridge University Press; 2004. ISBN 0521813972.

[6] Ananiadou, S., Thompson, P., Thomas, J., Mu, T., Oliver, S., Rickinson, M., et al. Supporting the education evidence portal via text mining. Philosophical Transcations of the Royal Society A 2010;368(1925):3829–3844.

[7] Salton, G., Wong, A., Yang, C.S.. A vector space model for automatic indexing. Commun ACM 1975;18(11):613–620.

[8] Ding, Y., Yi, K., Xiang, R.. Design of paper duplicate detection system based on lucene. Wearable Computing Systems, Asia-Pacific Conference on 2010;:36–39.

[9] Frantzi, K.T., Ananiadou, S., Mima, H.. Automatic recognition of multi-word terms: the c-value/nc-value method. Int J on Digital Libraries 2000;3(2):115–130.

[10] Okazaki, N., Ananiadou, S.. Building an abbreviation dictionary using a term recognition approach. Bioinformatics 2006;22:3089–3095.

[11] Bhagat, J., Tanoh, F., Nzuobontane, E., Laurent, T., Orlowski, J., Roos, M., et al. BioCatalogue: a universal catalogue of web services for the life sciences. Nucleic Acids Research 2010;38:689–694.

[12] Kano, Y., Dobson, P., Nakanishi, M., ichi Tsujii, J., Ananiadou, S.. Text mining meets workflow: linking u-compare with taverna. Bioinformatics 2010;26(19):2486–2487.

[13] Hao, Y., Zhang, Y., Cao, J.. Web services discovery and rank: An information retrieval approach. Future Generation Computer Systems 2010;26(8):1053 – 1062.

[14] Zhuge, H., Li, Y.. Semantic profile-based document logistics for cooperative research. Future Generation Computer Systems 2004;20(1):47 – 60.

[15] Altintas, I., Berkley, C., Jaeger, E., Jones, M., Ludascher, B., Mock, S.. Kepler: an extensible system for design and execution of scientific workflows. In: Scientific and Statistical Database Management, 2004. Proceedings. 16th International Conference on. 2004, p. 423–424.

[16] Hull, D., Wolstencroft, K., Stevens, R., Goble, C., Pocock, M., Li, P., et al. Taverna: a tool for building and running workflows of services. Nucleic Acids Research 2006;34:729–732.

[17] van der Aalst, W., ter Hofstede, A.. YAWL: Yet another workflow language. Information Systems 2005;30(4):245–275.

[18] Lathem, J., Gomadam, K., Sheth, A.P.. SA-REST and (S)mashups: Adding semantics to RESTful services. International Conference on Semantic Computing 2007;11(6):469–476.

[19] Sheth, A.P., Gomadam, K., Lathem, J.. SA-REST: Semantically interoperable and easier-to-use services and mashups. IEEE Internet Computing 2007;11(6):91–94.

[20] Fielding, R.T., Taylor, R.N.. Principled design of the modern web architecture. ACM Transactions on Internet Technology 2002;2:115–150.

[21] De Roure, D., Goble, C., Stevens, R.. The design and realisation of the virtual research environment for social sharing of workflows. Future Generation Computer Systems 2009;25(5):561–567.

[22] Elmroth, E., Hernández, F., Tordsson, J.. Three fundamental dimensions of scientific workflow interoperability: Model of computation, language, and execution environment. Future Gener Comput Syst 2010;26:245–256.

[23] Dean, J., Ghemawat, S.. Mapreduce: simplified data processing on large clusters. Commun ACM 2008;51:107–113.