School of Electronics and Computer Science
Faculty of Engineering, Sciences and Mathematics
University of Southampton

Timothy Lewy

9th May 2006

A Consistent Reference Service for the Interoperation of EPrint Repositories

Project supervisor: Eric Cooke
Second examiner: Adam Prügel-Bennett

A project report submitted for the award of
Computer Science (MEng)

Abstract

Current Institutional Repository packages do a poor job of maintaining the article's metadata in a consistent fashion. Documents and other entities are unreliably identified and there exists no mechanism for correlating related data between multiple repositories. A consistent reference service (CRS) mediates and maps between different identifiers, from multiple sources. It overcomes the shortcomings of packages such as EPrints and allows the construction of useful applications and services, such as automatic CV generation or citation impact profiling. This project has developed a highly efficient and scalable CRS, capable of tracking many thousand identifiers. It utilises semantic web technologies to remain open and responsive, providing intuitive and flexible services for searching and retrieving information. A sophisticated plug-in for the EPrints software has been developed, which utilises the CRS to improve the inherent consistency of the metadata; reinforce the use of local naming schemes and significantly enhance the repository's user interface. A CRS deployment is already in active use by researchers of the ReSIST Project.

Contents

Acknowledgements

Many thanks to the AKT team, especially Hugh Glaser, for ideas, expertise and resource; to the ReSIST project for their interest and collaboration; to the EPrints team, especially Chris Gutteridge, for technical help and support with the EPrints software; to Les Carr for his support and for providing the development testbed; and to Eric Cooke and Adam Prügel-Bennett for their continued support and guidance.

1 Introduction

Since it was proposed that the UK Research Council (RCUK) mandate that all council funded research must be made available through institutional repositories (IRs) [1], Institutions have made considerable investments in software such as GNU EPrints [2] or MIT DSpace [3]. This switch to open archiving has resulted in a rapid influx of, potentially, very useful metadata, into the public domain. However the current available repository packages do not promote or provide effective mechanisms for keeping data reliable and consistent, especially when considering the use of data from multiple repositories.

1.1 Problem

Within institutional repositories there is data stored regarding many different entities; not just documents, but anything that is referenced by a document's metadata, such as authors, institutions, funding councils and journals, to name but a few. Ideally one would be able to obtain a complete and consistent set of data about any given entity: this would provide an invaluable service, upon which could be built all manner of applications. It is easy to envisage being able to automatically generate CVs, or at a glance see the citation impact of a particular project or group. This is not currently possible, as repositories only hold canonical representations of the documents. Links from documents to other types of entities are left as free floating textual references, or use unreliable identifiers such as email addresses. This makes it difficult to collate data on a specific entity within a single repository, let alone between multiple repositories.

Whilst one could simply perform a text search, there would be no way to tell that "Wendy Hall" is the same person as "Hall, W" or as a "Wendy Hall" in another repository. By analysing the list of researchers' names in the RAE 2001 returns [4, 5] the extent of this problem becomes clear: 10% of names lead to a clash between two or more individuals. So, for 1 in 10 people a free text search will return references not only to them, but to one or more others as well.

Some repositories, such as EPrints have the facility to use identifiers for some specific entity types, such as authors and editors. This identifier is generally an optional field on the document deposition form. It is up to the individual repository administrator to choose a format and ensure it is used consistently. Frequently existing departmental ids or email addresses are used. If used properly, these identifiers can resolve the problem of local linking between the supported entities, by providing a unique identifier with which entities can be linked and referenced. However id fields are frequently left blank or worse are completed incorrectly.

EPrints is able to make these identifiers externally available; but even if the identifiers are used effectively internally, they are only unique locally and each IR employs its own conventions for assigning them. Therefore any interested party, wishing to gather data on entities between different repositories, would have to manually map local identifiers in one repository to those in another.

It would not be feasible to propose any scheme or naming authority for providing globally unique identifiers for every entity, in every repository. Such a system would restrict the open growth of online academic communities and would prove infeasible to implement or enforce. Any proposed conventions for unique identification would inevitably leave parties or individuals (such as foreign bodies or institutions) out of its scope and would therefore fail to provide truly, globally, unique identifiers.

A system is required that can serve to ensure that identifiers are used both accurately and consistently within repositories, and that can mediate between the diverse array of existing referencing systems. It should identify and provide suitable canonical references for those entities which are not supplied local identifiers.

1.2 Goals

The goal of this project is to design and develop a consistent reference service (CRS). This will aid cross-referencing across repositories, by recording and making available, mappings between equivalent local identifiers.

The system is required to provide several areas of functionality:

- Services, which can be used by any systems or individuals, to obtain identifiers for specific entities. It should be able to perform this function either given any single identifier, or by somehow searching for it.

- Services for adding, removing and maintaining identifiers and mappings.

- An interface for administrators or other authorised users to manually map between different entity's identifiers.

Plug-ins for EPrints can then be developed to utilise the CRS in order to allow users to more easily and correctly identify entities and their details when depositing new documents. This should significantly increase the consistency and reliability of the repository's internal linking. The plug-in would also aid the user in searching and exploring the repository, by providing the facility to utilise the canonically referenced and matched entities when performing text searches. This would yield far more accurate, error free, results when searching for publications relating to a specific entity.

Once established, the services provided by the system must enable inter-repository data sharing and efficient utilisation of extracted metadata; thus allowing third parties to construct services such as automatic CV generation. The CRS must be highly scalable, readily deployable and reliable. The plug-in to EPrints must be seamless; it should increase the consistency of EPrint's internal linking, whilst significantly easing and benefiting user interaction.

2 Background

2.1 Data Harvesting

In order for third party applications to utilise the metadata stored in repositories there must be an open and standardised manner for obtaining the data. The Open Archives Initiative (OAI), one of the driving forces behind the shift towards open access to research [6], has done significant work in this area. They have developed a standard protocol, OAI-PMH (see [7]), for the harvesting of metadata from compatible repositories. Using this protocol, which is now supplied with most IR packages as standard, any interested party may harvest information on the documents stored within.

The OAI-PMH protocol generates metadata, adhering to a given standard (unqualified Dublin Core [8] by default), based upon the information made available by the repository. Any program fluent in Dublin Core can utilise the interface. The OAI protocol simply makes the internal identifiers for entities available externally; thus it is still down to any system using the protocol to make sense of the data and to map between identifiers.

Some services have already been built upon the OAI protocol as it is. These services must either make do with very inconsistent data, or must perform the entire mapping process themselves. Citebase is one example of such a service [9]. It provides a citation based search engine for documents within its range of repositories. Citebase attempts to automatically link citation references to their document instances. However, performing this process from scratch, for a single application, is a laborious process and once complete there is no way for others to take advantage of the findings.

2.2 The Semantic Web

A great deal of research has been performed on the use and storage of metadata on the internet. One of the most significant outcomes has been the development of the semantic web.

The current web is designed solely to be viewed and understood by human users; the information is marked up in a way that, once interpreted, makes the pages easier for users to view and browse. It is very difficult for a computer system to discern useful information from the web. The semantic web is a layer of metadata added on top of the existing web, designed to allow machines or programs to navigate and process the information more easily. It achieves this using metadata described by mark-up languages such as RDF(S) [10] and OWL [11], which are extensions of XML. It is a semantic network of knowledge, or information, rather than web sites. Instead of pages connected by hyperlinks, there are resources (the objects of the knowledge e.g. a person), connected to one another by predicates. For example Tom (resource) is the brother of (predicate) Sam (resource). The predicates themselves are in fact also resources, which may have other connections to them. This subject-predicate-object structure is known as a triple or statement. So that resources can be identified and referenced, they are given identifiers, which are URIs similar to the URLs used to locate web pages.

The semantic web allows applications to be developed that take advantage of the information and services available on the internet without human intervention. Simple examples include automatic buying agents and intelligent search services. However the implications of the semantic web are more far reaching. Ultimately the semantic web could enable a hitherto unseen integration and pervasiveness of knowledge technologies in everyday life. It would allow semantic agents to perform complex tasks on the behalf of their user, with little or no interaction. Tim Berners-Lee gave an excellent example of this in [12]. In his example, a semantic web agent is used not only to schedule doctors appointments at times to fit several people, but also to find available reputable providers of the prescribed treatment, within an acceptable distance.

The OAI-PMH protocol generates metadata that is very easily converted into a semantically compatible form. This allows semantic web applications to be developed that make use of the data stored within Institutional repositories. The aforementioned example applications, such as automatic citation impact profiling, are therefore examples of semantic web applications.

Converting metadata from repositories into a semantic form does not immediately resolve the problem of reconciling different identifiers. In fact, it is an inherent and crucial problem within the semantic web that represents a significant impediment to the development of many semantic web applications. However, this existing compatibility means the technologies and techniques that have been developed for the semantic web can be utilised to resolve the problem for Institutional Repositories.

2.3 The Matching Process

The job of a mapping process is to identify references or identifiers that represent the same entity. This issue is neither new nor native to the semantic web, or even Computer Science. It is prevalent in many items of classic literature. As was explored in [13], the great Roman orator and statesman know as Cicero was frequently referred to as Tully, which is a confusing case of coreference (coreference is when two references refer to a common entity). Another, Marcus Porcius Cato, had a great grandson of the same name, both of whom were senators. It is understandably hard to distinguish the two and to identify which texts refer to the same person.

The problem has been encountered in the fields of Natural language processing and AI. For language processing, the problem is the same as that above. In text it is generally easier to resolve; there is more information to go on: the context that the reference appears in can be utilised and combined with heuristics to determine coreference. In AI, the approach largely taken is to use the *unique name assumption* [14], whereby a 1 to 1 relationship between resources and identifiers is enforced, thus avoiding the problem. This is, unfortunately, not transferable to this project as it would be impossible to implement without using a naming authority.

Within the semantic web, some approaches to tackling inconsistent references have been proposed. Many of these, such as Ontocopi [15], which utilises communities of practise [16], attempt to automatically resolve matching references and remove duplicates, based upon available metadata. However, such solutions do not pick up every coreference and only work well under an ideal environment (i.e. where there is a lot of metadata). One, sophisticated, system for automatically resolving coreference has been developed by the University of Washington (see [17]). It operates by gathering positive and negative

evidence for potential resolutions, much like other systems. However it is notable, as it takes into consideration factors that others do not: It considers the impact of making a resolution, looking at new evidence that would be created if the resolution were carried out. It also compares the values of different types of the resource's attributes to one another for similarity (for example, looking for names within email addresses). These extra considerations mean it is capable of finding matches between many references that have little metadata attached.

Resolving coreference automatically is a difficult and potentially risky process. If two references that are resolved to be the same are in fact not, the data regarding the two separate resources will be merged and become incorrect. Additionally, one incorrect resolution could provide false evidence that indicates other references should be reconciled, creating a cascade affect. This makes it all the more crucial that when a mapping process is performed accurately, the results are recorded and made available for the use of others.

2.4 Previous Work

2.4.1 Expressing Coreference

The semantic web has an existing, rudimentary method for identifying coreference. Information is added to the metadata, recording any equivalence between references. This is incorporated into the Web Ontology Language (OWL) [11], as a set of predicates. The predicate 'owl:sameAs' asserts that two references are equivalent, whilst 'owl:differentFrom' asserts that two references are not equivalent. The main advantage of these predicates is that they may be utilised by the inference mechanisms built into modern semantic data stores [18]. However, they only give a one-way association (subject -> is the same as -> object) and only a 1 to 1 cardinality. The result of this can be the creation of awkward networks, or graphs (see Figure 1). This structure presents several disadvantages. In order to be able to efficiently retrieve the entire graph, given any one of its nodes, the graph must be complete (i.e. every node connected to every other node). Achieving this would result in the number of triples required, increasing exponentially with the number of nodes. OWL also statements attach potentially undesirable semantics. Asserting that two resources are 'owl:sameAs' is a very strong statement, especially when bearing in mind that reference resolution is a tricky process, occasionally requiring equivalences to be revoked. Such connotations are confusing and obscure the genuine metadata. It is best to keep the resolution process separate, so that it can be treated as a separate exercise.

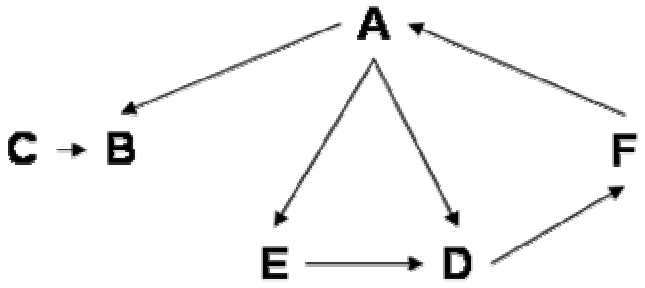


Figure 1. An example owl:sameAs graph. Sourced from [17].

These disadvantages are addressed by [19], in which I have proposed a methodology and ontology whereby equivalent references (duplicates), are grouped into sets, or 'bundles',

using RDF. Each equivalent reference is identified as being a member of the bundle, with a single triple that links back to the bundle entity. Each bundle is stored within a separate graph, allowing them to be individually removed, asserted or communicated, maintaining a level of separation from the other metadata. This makes far more efficient triple use and is preferable, as given any one of the references, it is easy to obtain all the rest, unlike when attempting to traverse a graph created using OWL predicates. It also allows the use of simple set theory upon the equivalences. For further details and examples of the specific structure and syntax of bundles, see Appendix A.

2.4.2 ACIS Project

The Academic Contributor Information System (ACIS) [20] is a system that is being developed to aid academics in maintaining an online profile and CV. The system has a sizeable overlap with this project; it represents an alternative means of achieving some of the goals, namely online CV generation.

ACIS harvests information from EPrints repositories, with the aid of a purpose built plug-in [21], which generates metadata whenever the repository is updated. It stores this data in its own database.

The onus for performing linking and coreference resolution is placed entirely on the academic themselves. If they wish to participate and maintain a profile, they must register with the service and provide basic metadata on themselves [22]. From this metadata the system performs heuristic text searches for matching documents and institutions; the user is presented with a list of possible matches and is asked to select ones that relate to them. Selected items are added to the user's metadata.

ACIS then utilises its own author identification plug-in to keep the profile up to date. When depositing documents into a repository, on entering author details, the depositor is presented with a list of matching authors present in the ACIS knowledge base. If the depositor chooses one, the document is added to the author's profile.

All linking of documents to authors is kept within the ACIS database; there is little or no emphasis on improving linking within repositories. Each document has to be individually linked to its author.

3 Solution

Different processes have been proposed for automatically mapping between different references, however they tend to be bespoke systems, restricted to specific domains. Rather than attempting to develop a universal mapping process, it would be far more useful to provide a service that can record the results of each separate process and use them to make a consistent set of references for any resource, available from a single source.

Such a consistent reference service keeps a permanent, accessible, record of the mappings between references. It provides interfaces that allow one to upload established mappings or to perform the process manually. It serves as a medium through which different parties communicate, by supplying recorded mappings 'on tap' to any interested application, as and when required (see Figure 2). The CRS must be capable of keeping track of thousands of different references and equivalences. If it is inefficient or does not scale well, it will not provide sufficiently responsive services in a live deployment.

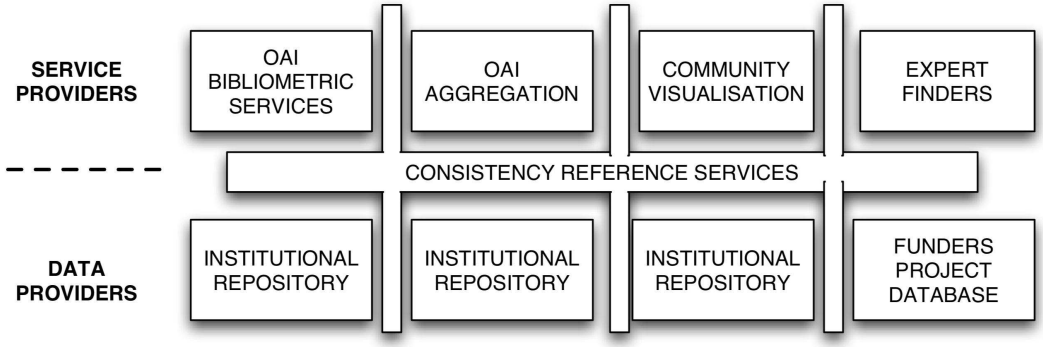


Figure 2. The Consistent Reference Service acts as a layer between other services, through which they can communicate. Sourced from [5].

From EPrint's perspective, identifiers can be uploaded to the CRS from multiple repositories. There, they will be mapped either manually by an administrator, or by an automatic process. The mapped references can then be used to gain greater returns. They could be used to build services that cross-reference between repositories, or to provide user aids, such as an auto complete utility for the document deposit form. By combining the CRS with a plug-in that suggests identifiers for entities in documents being deposited, the number of new references being created is reduced, limiting the amount of mapping that needs to be done in the CRS.

Along with the references, metadata can be stored to allow them to be searched for. Also, by recording where each reference has come from, systems can better choose a reference that is appropriate to them. For example, an EPrints repository could choose to always use a native reference, where available, rather than a foreign one. For EPrints, this represents a far more intuitive solution for automatic CV generation than the ACIS system. Rather than laboriously linking each document from every repository to an external profile, the use of existing local identifiers is reinforced (using an auto complete

plug-in). This improves the internal level of consistency and linking between entities; making it a simple matter of cross-referencing the identifiers externally (using the CRS). Once done, documents for a given author can be aggregated from each IR and presented in CV form.

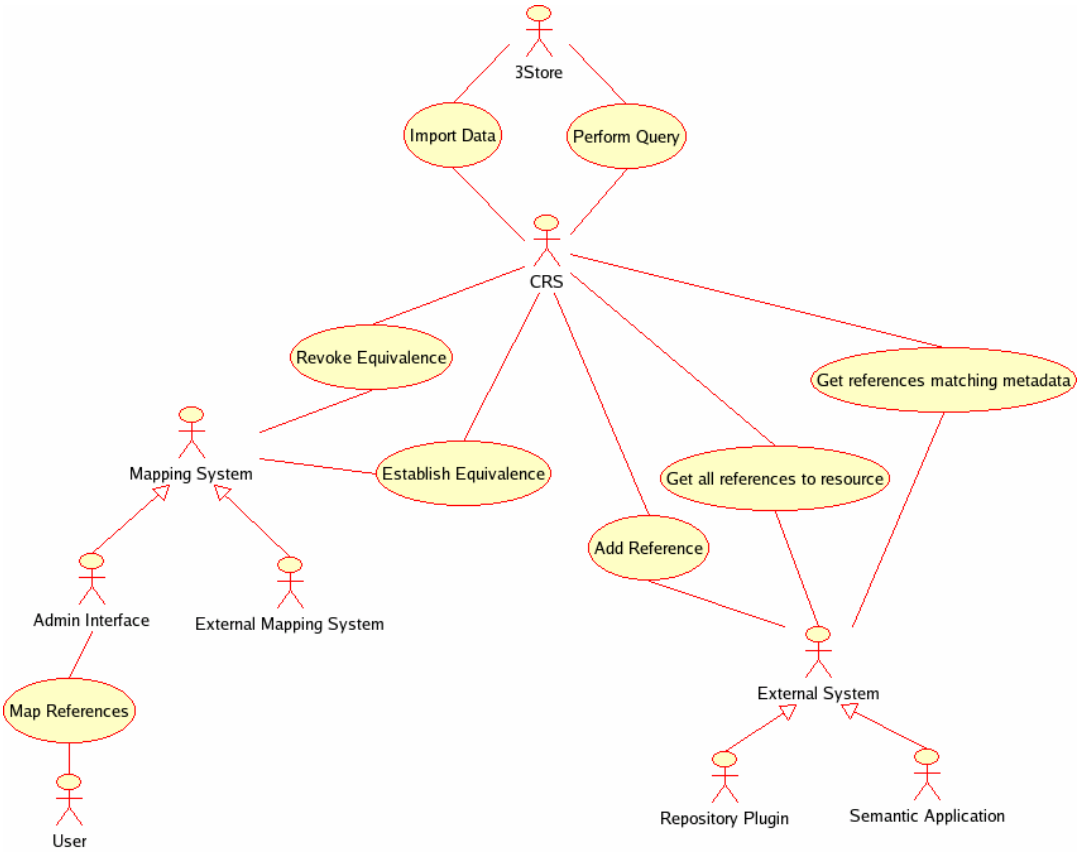


Figure 3. CRS Use Case Diagram.

There are essentially two classes of stakeholders for a CRS; these are shown on the left and right hand sides of Figure 3. The actors on the right are those that wish to use and gain from the system. They could be individuals, plug-ins for software such as EPrints or other semantic applications (some examples of these stakeholders are displayed in Figure 2). The actors on the left are those that contribute to the system, by aiding in the mapping process. These could be administrators performing the task manually or automatic mapping systems uploading their results.

Clearly it would be infeasible for a single CRS to serve every repository, in every field. A CRS should be used to maintain consistent references within a specific community, such as UK computer science or semantic web research. It is not necessary to try and provide services for a very wide community; as references are rarely used across certain boundaries, like that between Computer Science and Theology. Even on the odd occasion when they are, it is far less crucial for the references to be consistent.

3.1 Alternatives

It would be possible to achieve some of the objectives of this project by other means. A system for suggesting identifiers or form values, when submitting documents, could be constructed, purely internally to EPrints, by searching for possible matches within the EPrints database. However, this would not be able to make suggestions for any entities that are foreign to the system. It would also not be able to facilitate any higher level of interoperability between systems.

It is also possible that another method could be found for allowing foreign repositories to interact. A naming authority could achieve this, but for the reasons discussed previously (See section 1.1), this would not be feasible. A system that attempts to automatically match identifiers between repositories, as and when required, could also be conceived. Unfortunately, as desirable as this would be, the inevitably poor accuracy of such a system would make it, at best, very unreliable and at worst positively dangerous.

This solution currently provides the most complete and reliable fulfilment of the project requirements.

4 System Design

The system is broken down in to three subsystems: The CRS server, which is the backend of the system, keeps track of references and provides services for using and maintaining them; the administration interface, which allows a user to establish equivalences between references manually; and lastly, the plug-in to EPrints, which utilises the CRS to improve linking within the repository by aiding document deposition and search. The diagram in Figure 4, on the next page, shows the layout of the system.

4.1 Overview

In order to be open and compatible, the system is deployed onto the web. It is, after all, essentially a semantic web application and therefore any parties wishing to interface with it, would expect to use an open web interface, or service. This also allows for a lightweight implementation of the EPrints plug-in and Administration interface: both can operate by parsing XML data from the CRS server.

The database behind the CRS is a 3store semantic knowledge base [18]. This allows the system to be flexible, whilst remaining highly efficient. The 3store software's response time has been proven to scale linearly with the amount of data stored. Knowledge bases, instead of holding the data as columns and tuples, store it as a semantic network. This is very convenient, as the system is required to hold undefined networks of metadata and relations between references. It is highly flexible and allows the utilisation of inference mechanisms and the advanced capabilities of semantic query languages such as SPARQL [23]. This means, otherwise very complicated transactions with the system, can be carried out frequently with a single query. If a relational database were used, the system would have to adhere to a specific table schema, which would not allow for the constantly changing and evolving data that comes from an institutional repository. It would also be much more difficult to store and retrieve XML metadata, lacking the convenient compatibility that knowledge bases have with the semantic web.

The CRS server is written in PHP, as this allows for rapid deployment and the fastest and easiest connectivity with the web. It connects to the database using purpose built PHP API classes, which perform queries and assert data, via 3Store's system tools. The CRS makes itself available over the web via simple web services, which accept HTTP POST or GET requests, and return XML results in the response body. Each service has a URL and an endpoint script, which calls the relevant functions from the central CRS classes.

By using such lightweight web services, it is very easy for data obtained from the CRS to be integrated into any other application. The only requirement for use is a web connection. This is demonstrated in the design of the EPrints plug-in, which uses AJAX [26] techniques to pull XML in from the CRS dynamically, without having to reload the page. The plug-in, on loading a page within an EPrints repository, attaches itself to any fields that it identifies as being suitable for enhancement. Then, when the user types in the field, it makes a connection with the CRS and searches for references matching the entered text. If references are found, the user is asked whether they wish to fill the form with suitable metadata.

Data is uploaded to the CRS from EPrints via a simple PERL script that exports any entities that are not already present; again this is achieved via the web services.

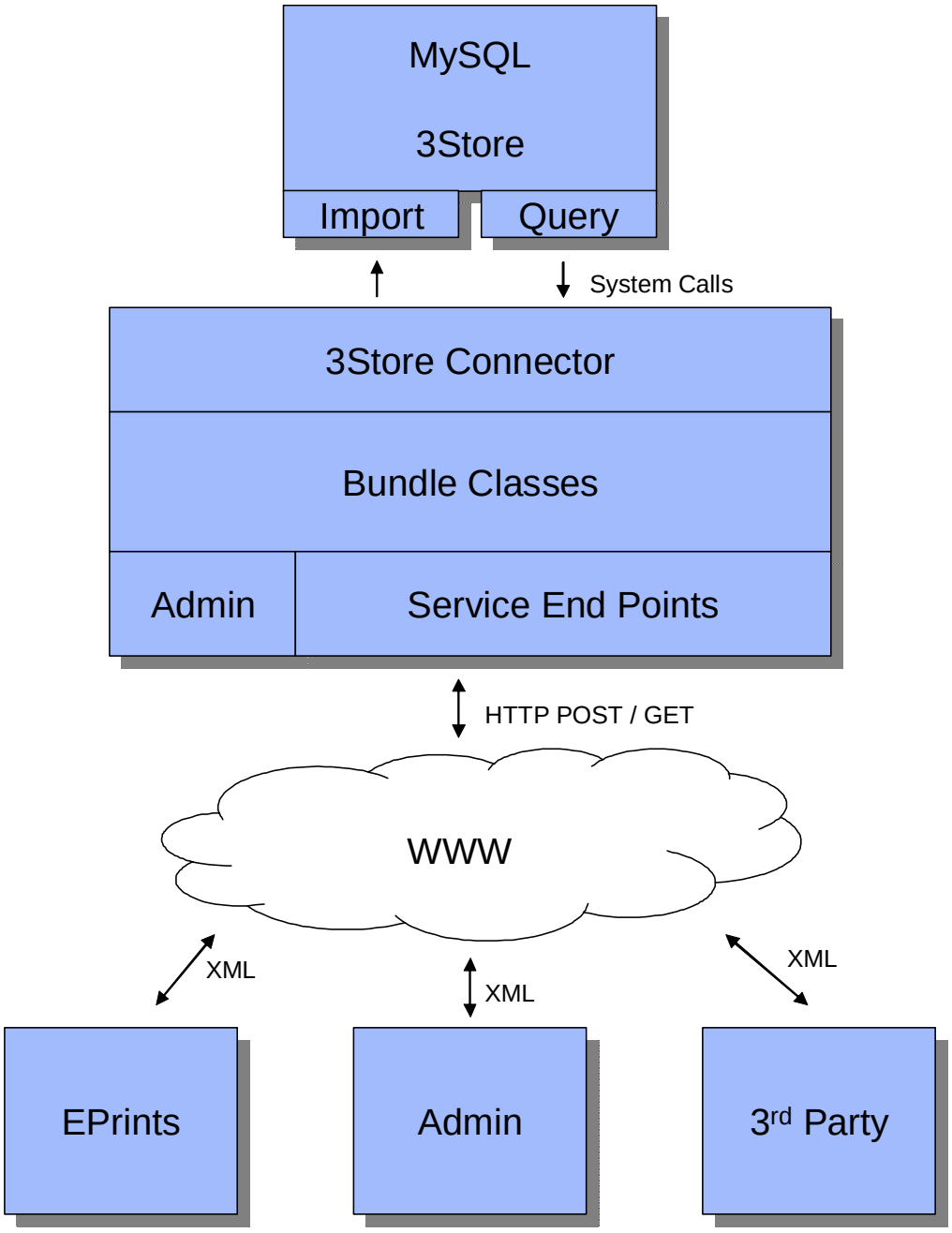


Figure 4. System Overview

4.2 Implementation Notes

Initially both the backend and the extensions to EPrints were developed on the same computer. Once development was completed, the system was rolled out on a large 3rd party server, in order to facilitate the deployment and scalability testing. See Appendix B for the implementation schedule.

5 CRS Design and Implementation

The consistent reference service's main technical challenge is the ability to keep track of virtually any number of references and equivalences. This is achieved by using a well thought out database design and backend system. Though the rest of the system has been built around the GNU EPrints software, the CRS has been deliberately made software independent, allowing it to be used with any software package.

5.1 Database Structure

The CRS is built upon version 3 of the 3Store software. 3Store is a triple store implementation that sits atop of MySQL. Queries made to it are translated into SQL queries upon the underlying relational database. The advantage of this approach is that it inherits the mature query optimisation present in MySQL, helping it to maintain a high level of responsiveness and scalability.

Triple stores hold semantic networks, composed of statements, or triples (thus the name). As has been discussed in a previous section, a triple represents a subject – predicate – object relation. Multiple, interrelated, triples create graphs of related data. In order for graphs to express useful information, they must adhere to a known structure. This is known as the ontology and can be specified externally to the triple store, using a mark-up language such as OWL or RDFS [24]. Section 2.4.1 explored the difference between two alternative graph structures, or ontologies, for expressing coreference, the decision between them and the efficiently of the system are inexorably linked. Through the work done on reference bundling in [19], an extremely efficient ontology has already been developed that can be easily adapted and extended for the CRS.

3Store only allows the assertion or removal of entire, uniquely named, graphs at a time. Thus, not only is the structure of the data important, but also the organisation and division into individual graphs. It would be undesirable to have to reassert all the metadata for a reference, every time its equivalences are updated. Therefore, metadata and bundles should be very carefully divided into separate graphs. This also allows searches to be performed through the metadata, without having to search through the data comprising the bundles as well.

Figure 5 shows the typical structure of a bundle. In the diagram, the URIs are shown without namespaces. The namespace used is http://www.ecs.soton.ac.uk/~tml203/CRS. Each bundle represents a single entity within the system. References to the entity are related to a central bundle resource. One reference from each bundle is designated as canonical, which is an entity selected to be recommended for people to use. It is simply selected lexicographically, as the choice of canon is irrelevant, as long as it remains constant, whilst the bundle is constant. One additional piece of metadata is recorded: the date on which the bundle was last inserted, this allows bundles to be sorted by most recent activity.

As no reference can appear in more than one bundle (such a situation would not make sense: two bundles with a common reference must represent the same entity), the canon is a convenient means of uniquely identifying the bundle. The Bundle URIs are

constructed by appending 'http://www.ecs.soton.ac.uk/~tml203/CRS#bundle-' to the MD5 hash of the canon's URI. Using a hash ensures the removal of any special characters and shortens the URI to a more appropriate length. The graph containing the bundle is identified by the same URI as the bundle itself.

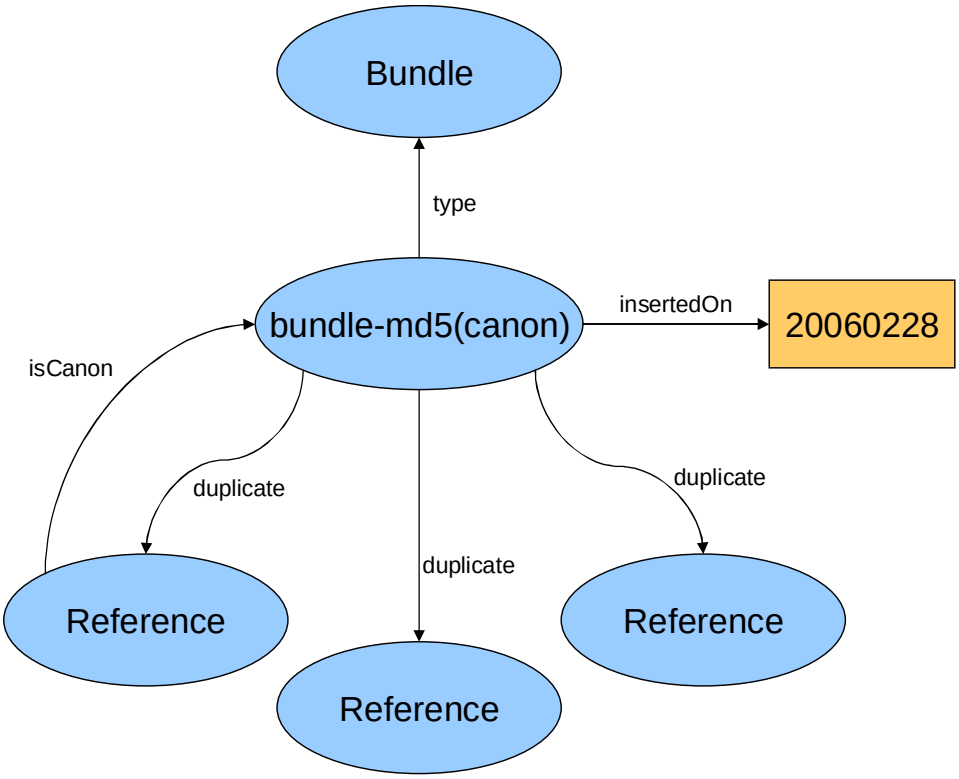


Figure 5. Bundle Graph Structure.

Separate to the graphs containing the bundles, is the rest of the metadata. This too is divided into separate graphs, this time with one graph for each reference's metadata. By dividing the metadata according to the reference it is related to, one can efficiently add, update or remove a single reference. The metadata's graph is named similarly to that of the bundles, only using the prefix 'metadata-' and the MD5 of the reference's URI. The amount and type of metadata that can be stored is not restricted. However, as shown in Figure 6, for complete functionality, each reference should have four specific pieces of information available: An rdfs:label attribute, so the reference can be displayed with a name; The URI of the reference's origin (an EPrint repository OAI id for example); A string of keywords called the search string; and a type attribute, which corresponds to the rdfs:Class that the entity is an instance of ('Creator' or 'EPrint' for example).

The type attribute is handled specially by the system. As different sources may use different types to represent the same thing (for example, a creator in one EPrint repository could mean the same thing as an author in another) the different types are added into bundles, much like normal entities. This allows them to be matched and combined, when searching for an entity of a specific type. The search string is a single index of keywords, in a specific place, for the CRS to perform searches on. If the CRS had to search through every item of metadata, for every reference, when performing keyword searches, it would be very inefficient.

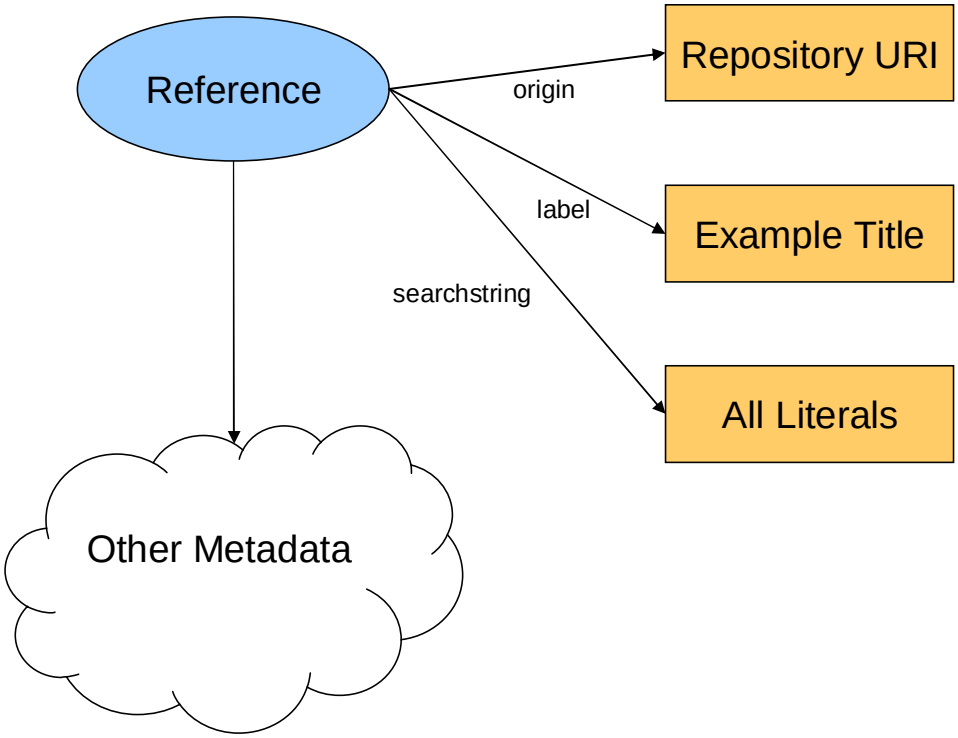


Figure 6. Metadata Graph Structure.

5.1.1 SPARQL Queries

Equally as important as the structure of the graphs are the queries that are used to retrieve the data. SPARQL (Simple Protocol and RDF Query Language) is the query language employed by the latest version of 3Store. A basic query consists of a list of desired variables, followed by one or more triple clauses. Filters, limits and order clauses can also be appended. For a full description of the SPARQL syntax, see [23]. This section outlines the queries used by the system and how they work. Each query has been carefully optimised for the best possible response times. There are four main queries.

```
SELECT ?predicate ?duplicate
WHERE {
   <bundle-uri> ?predicate ?duplicate
}
```

Figure 7. Query to Obtain Bundle Contents Given URI.

The above query returns all the metadata associated with the bundle object. Two types of metadata are associated with bundles duplicates (the equivalent references) and insertedOn attributes; these are distinguished using the '?predicate' variable.

```
SELECT ?bundle
WHERE {
   ?bundle <http://www.ecs.soton.ac.uk/~tml203/CRS#duplicate> <reference-uri>
}
```

Figure 8. Query to Obtain Bundle Given a Reference.

The query in Figure 8 is used to obtain the URI of a bundle, given the URI of one of its references. Having obtained the bundle URI, the first query can be used to get the contents.

```
SELECT DISTINCT ?bundle
WHERE {
   ?bundle <http://www.ecs.soton.ac.uk/~tml203/CRS#duplicate> ?ref.
   ?bundle <http://www.ecs.soton.ac.uk/~tml203/CRS#insertedOn> ?date.
   ?ref <http://www.ecs.soton.ac.uk/~tml203/CRS#searchstring> ?string.

   ?typebundle <http://www.ecs.soton.ac.uk/~tml203/CRS#duplicate> <type-uri>.
   ?typebundle <http://www.ecs.soton.ac.uk/~tml203/CRS#duplicate> ?type.
   ?ref <rdfs:type> ?type.

   FILTER (
            regex(?string, 'search string') &&
            regex(?string, 'search string2')
   )
}
ORDER BY DESC(?date)
LIMIT 50
```

Figure 9. Query to Obtain Bundle from Metadata.

Figure 9 shows an example query for searching for a bundle by its metadata. The query is generated dynamically, when needed, and can vary depending upon the supplied parameters. The first three clauses within the WHERE statement are always present. These find the date on which the bundle was inserted, so that the results can be ordered and the searchstring indexes. The search is performed by filtering the returned bundles by a regular expression match on the searchstrings. Only bundles that have a reference, with a searchstring matching all the search terms, are returned. As each searchstring is every term that can be searched upon for that reference, concatenated into a single string, an 'all keywords' match can be performed by testing the string with a separate regular expression for each keyword.

The, optional, second block of clauses restricts the search to a specific type of entity. The type resources are all members of bundles, linked with equivalent types; the added clauses check that the bundles returned, contain a reference of a type equivalent to the one specified.

```
SELECT ?ref ?predicate ?object
WHERE {
   <bundle-uri> <http://www.ecs.soton.ac.uk/~tml203/CRS#duplicate> ?ref.
   ?model <rdfs:type> <http://www.ecs.soton.ac.uk/~tml203/CRS#metadata-model>.
   GRAPH ?model {
            ?ref ?predicate ?object
   }
}
```

Figure 10. Query for Obtaining Metadata for a Bundles References.

The last query, shown in Figure 10 obtains all the metadata associated with all the references in a given bundle. This is a relatively straightforward process; it simply finds

every triple associated with every reference. The search is restricting to only the metadata models; this prevents the bundle structure from being unnecessarily returned.

5.2 CRS Backend

On top of the 3Store, is a PHP system that interfaces with the database. It converts method invocations into query calls and result sets into objects. 3Store is supplied with several utilities for querying and asserting data. PHP interfaces with these utilities, using its built in 'system' function. Asserting is straightforward; the utility accepts plain RDF, along with flags to specify the model name and whether or not to flush the existing model. PHP is capable of producing RDF via the RDF API for PHP (RAP) [25]. The query utility accepts SPARQL queries and returns results in XML format, on the standard output. This is read by a custom-made parser class 'SPARQLer', which employs a SAX parser to return an associative array to the calling environment.

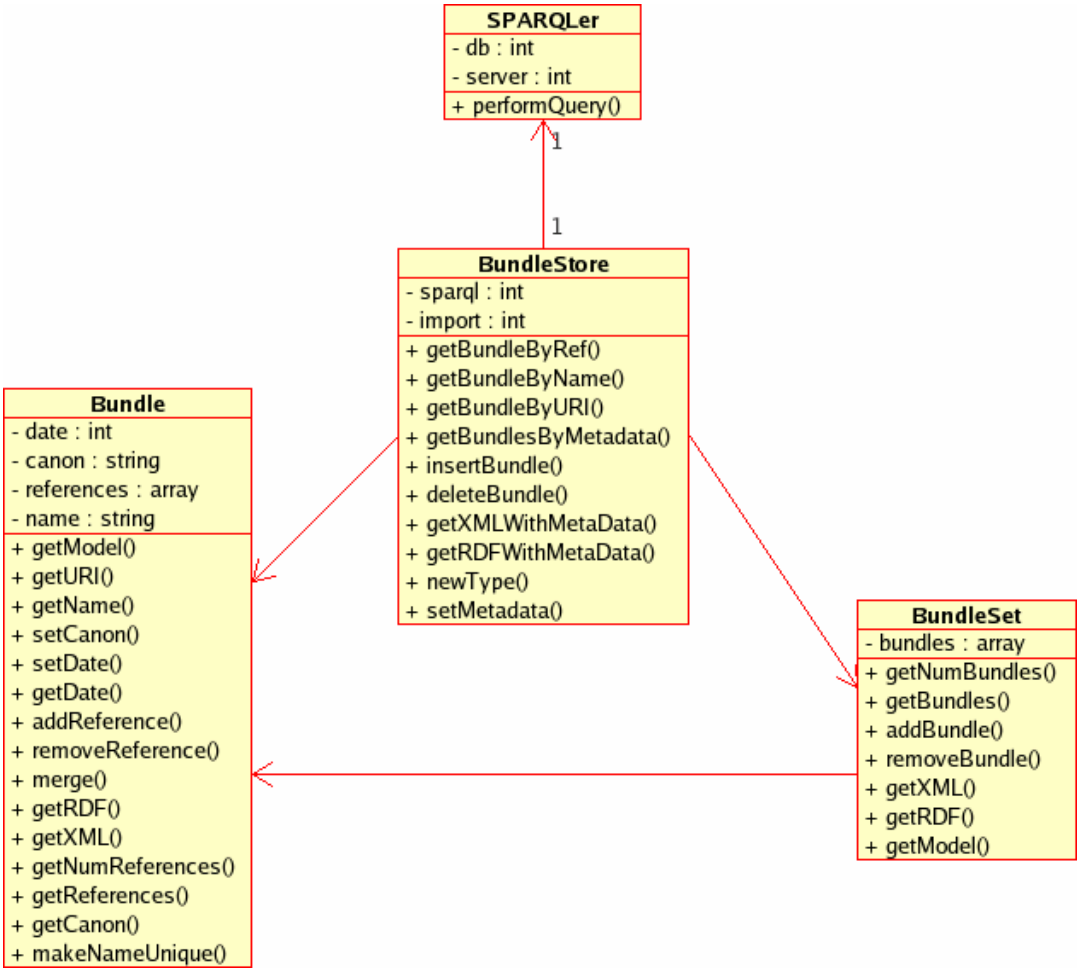


Figure 11. CRS Backend Class Diagram.

Rather than each web service implementing the required functionality itself, the basic operations upon the database and upon bundles are provided by a set of classes (see Figure 11). By providing this extra layer of abstraction, the amount of code reuse is increased, making debugging easier. There are three central classes: the BundleStore class, which provides functions on the knowledge base and utilises the SPARQL

queries; the Bundle class, which models a single bundle; and the BundleSet, which is a class for providing enhanced handling of multiple bundles. For details of the specification and implementation of these classes, see Appendix C.

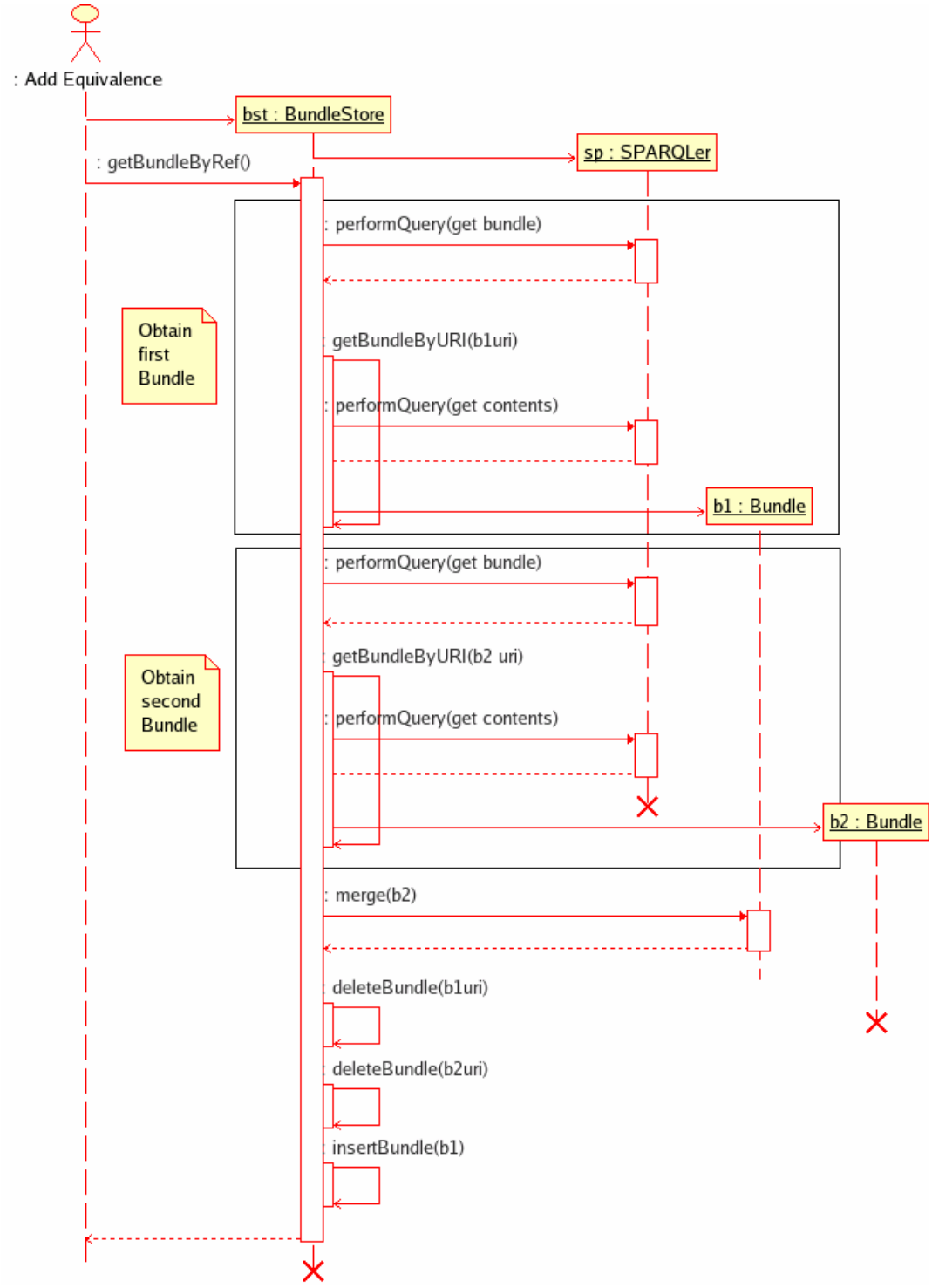


Figure 12. Sequence diagram for add equivalence function.

The sequence diagram in Figure 12 shows a typical interaction within the class system. It shows the action of establishing and recording an equivalence between two bundles in the knowledgebase.

5.3 CRS Services

The web services are very open and straightforward to use. They have well defined, clear functions, to allow easy integration with other systems. Requests are made using either HTTP POST or GET requests, variables passed to the service become the input parameters. The relevant SPARQL queries are called via the backend classes and the results are returned, in plain XML or RDF, like a webpage, to the requesting browser. The services are located within the root directory of the CRS server, addressable by the service name.

5.3.1 Add Reference / Remove Reference

The 'addreference' service is used for adding a new reference into the CRS. If the reference is not present in the CRS, a new bundle object is created and then inserted into the store. When inserting a reference, the label attribute must be included.

This service can optionally be used to set the metadata for the reference. The variables 'type', 'type1', 'type2' etc are used to set the type of the reference. Multiple types can be used in cases such as when an entity is both an EPrint and a Journal Article. If a new type is used, it is added to its own bundle and inserted. The type is also asserted to be a subclass of rdfs:Class, this allows it to be picked up by the knowledge base's inference mechanism as a resource class. A type may be inserted without a reference, if just the type variable is used. Other metadata can also be inserted by setting the variable corresponding to the predicate URI, to the value of the object.

addreference	
Variable	**Values**
reference	URI of the Reference.
type, type1… typen	Type(s) of entity being added. Non URIs supplied are appended to the CRS Namespace.
metadata	Any other variables are added as metadata. The name of the variable becomes the predicate. Non URIs supplied are appended to the CRS Namespace.
label	String to be used as label for the reference.

The 'rmreference' service removes a reference from the system. If the reference is in a bundle with other references, it is extracted and the rest of the bundle is left intact.

rmreference	
Variable	**Values**
reference	URI of the Reference.

5.3.2 Add Equivalence / Remove Equivalence

'addequivalence' asserts an equivalence between two references. This effectively merges the bundles that the references belong to.

addequivalence	
Variable	**Values**
reference1	URI of first Reference.
reference2	URI of second Reference.

The 'rmreference' service takes a single reference and removes all equivalences to it. It removes the reference from its current bundle and places it in one on its own.

rmequivalence	
Variable	**Values**
reference	URI of the Reference.

5.3.3 CRS by String / by Reference

There are two services for retrieving data from the CRS. The basic retrieval service is simply called 'crs'. This service takes a URI reference for an entity and returns all other references to that entity (the bundle).

crs	
Variable	**Values**
reference	URI of Reference.
format	Format of output: XML or RDF.
findmetadata	Include reference metadata: TRUE or FALSE

The second retrieval service is named 'crsbystring'. It is used for searching for an entity by matching the metadata attached to its references, against some list of keywords. It is also possible, though optional, to restrict the search to a specific type of entity.

crsbystring	
Variable	**Values**
string, string1… stringn	Keywords for searching
format	Format of output: XML or RDF.
findmetadata	Include reference metadata: TRUE or FALSE
type	Restrict search to type of reference. Non URIs supplied are appended to the CRS Namespace.

5.3.4 Ontology Exporter

As new types, or classes, are added to the system all the time, the knowledge base's ontology is dynamic, rather than static. This means it cannot be easily defined beforehand. To combat this, and provide a downloadable ontology for those who wish to use RDF from the CRS directly, a dynamic version of the ontology is made available. Located at '/ontology' on the CRS web server, it provides an OWL document that reflects the current state of the CRS ontology. Where multiple types have been asserted to be equivalent, owl:sameAs statements are included. The bundles themselves are not exported, as people wishing to employ the ontology should not need to have knowledge of bundles.

5.3.5 Entity Info Page

The entity info page, located at /admin/eninfo.php, takes the URI of a reference, in its 'reference' variable, as input. It displays the metadata relevant to that reference and also lists other entities that it is in some way connected to. This page is very useful as part of the mapping process, it provides extra details on an entity, allowing a more informed mapping decision to be made.

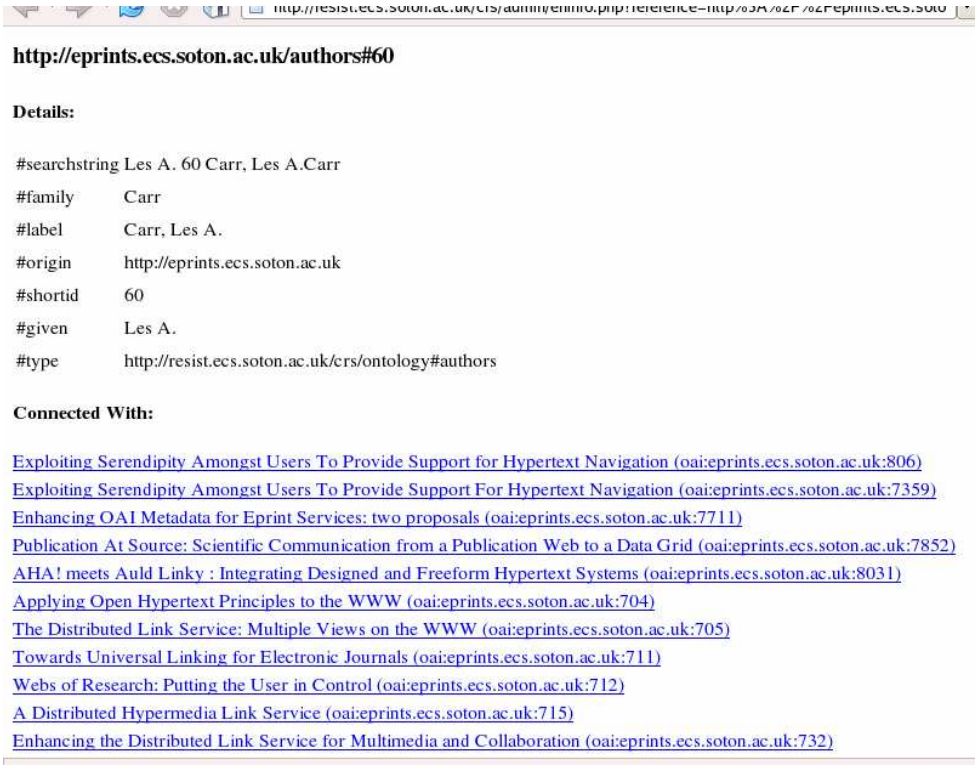


Figure 13. Screenshot of eninfo.php

5.3.6 Admin Interface

The administration interface provides a small system of web pages for performing the mapping process manually. Access to the interface is controlled through a login, which requires the user to specify the URI of the repository that they control. Each user is

restricted to only being able to assert equivalences where at least one of the bundles contains a reference from their repository.

The interface is broken into two sections. The first section, shown in Figure 14, is the type matching page. This page is to be used by a repository administrator to establish equivalences between the types of entities used in their repository and those that are already present in the CRS. The page displays a full list of every type in the system (even in a fully deployed CRS this should not become excessive). The administrator is invited to match equivalent types by selecting the appropriate checkboxes and clicking on the 'Match' button. This page also has the facility to manually add new types. Every reference displayed in the admin interface is provided with a link to its entity info page. Clicking on the link will open the info page in a new window.

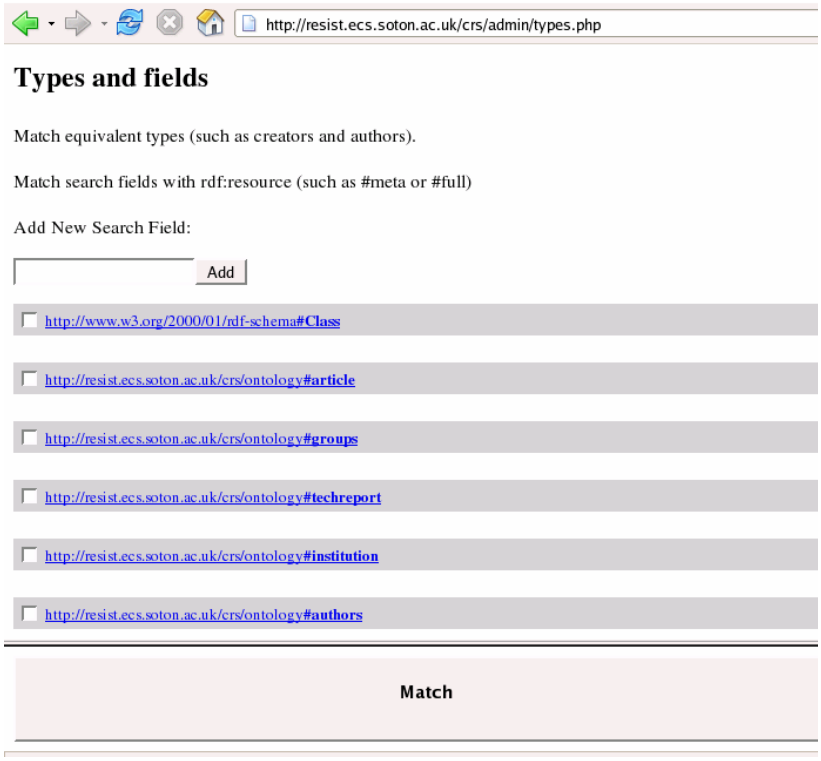


Figure 14. Screenshot of Type Matching Page.

The second half of this system is the entity matcher; this is the central matching system for manually mapping between entities. In order to establish equivalences between references, one must first be able to explore the contents of the CRS. This is done through the use of the search page, shown in Figure 15. This form allows the user to perform a keyword search, optionally restricted by entity type and repository of origin.

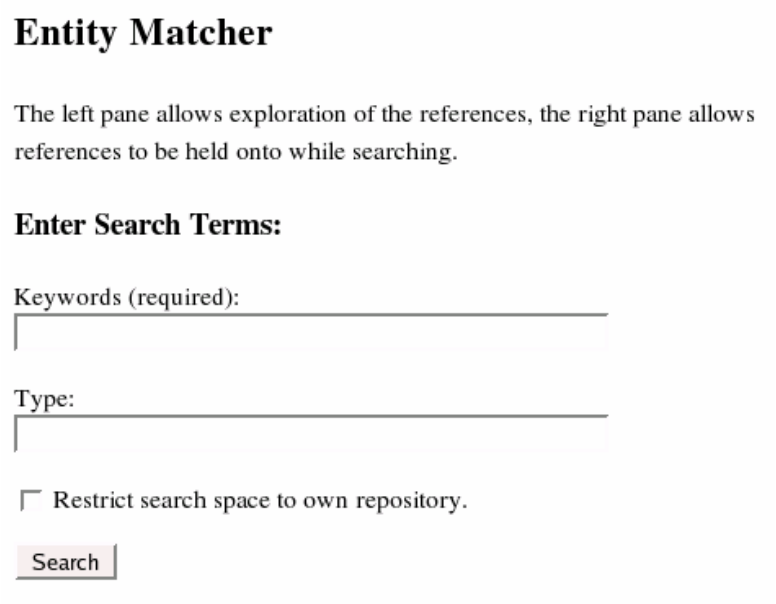


Figure 15. Screenshot of Search Fields.

Having submitted the search form, the page in Figure 16 is returned. The matcher works in a similar way to an online shop. There is an 'entity basket' on the right that the user can use to hold any interesting entities while they browse around (this is necessary as it is unlikely that every equivalent reference to an entity will be on the same page). On the left, the top 50 search results are displayed in alphabetical order. The idea is that the user browses through the entities in the repository, adding to the basket any references that they think might have duplicates.

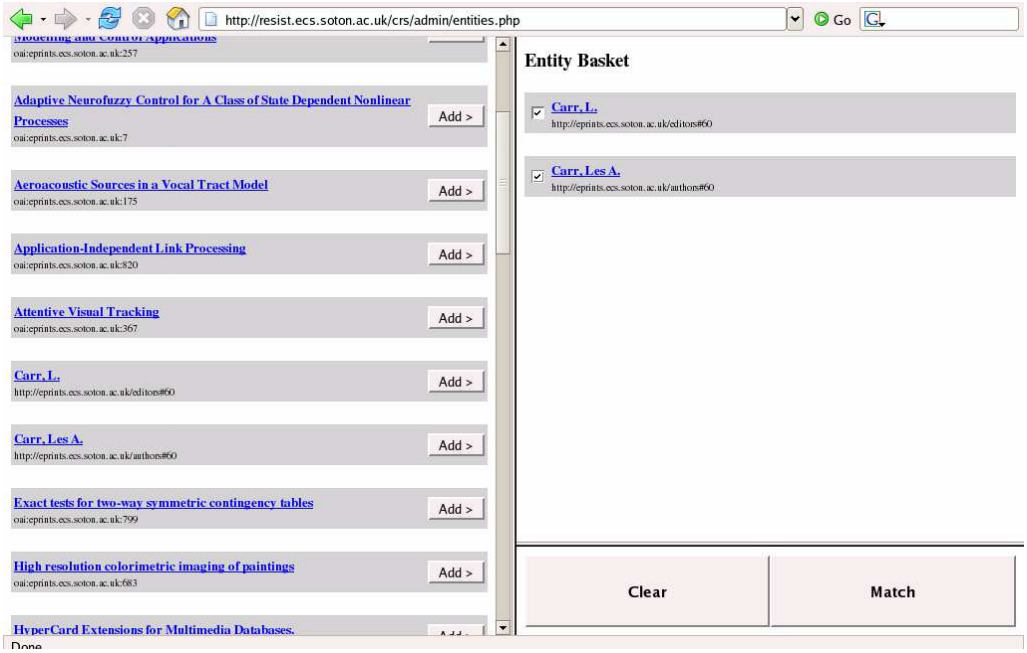


Figure 16. Screenshot of Entity Matcher.

When they are happy they have located all the duplicates, the references are matched using the checkboxes and 'Match' button. Figure 17 shows the two entities from the basket in Figure 16 having been matched; the listing on the left is also automatically updated.

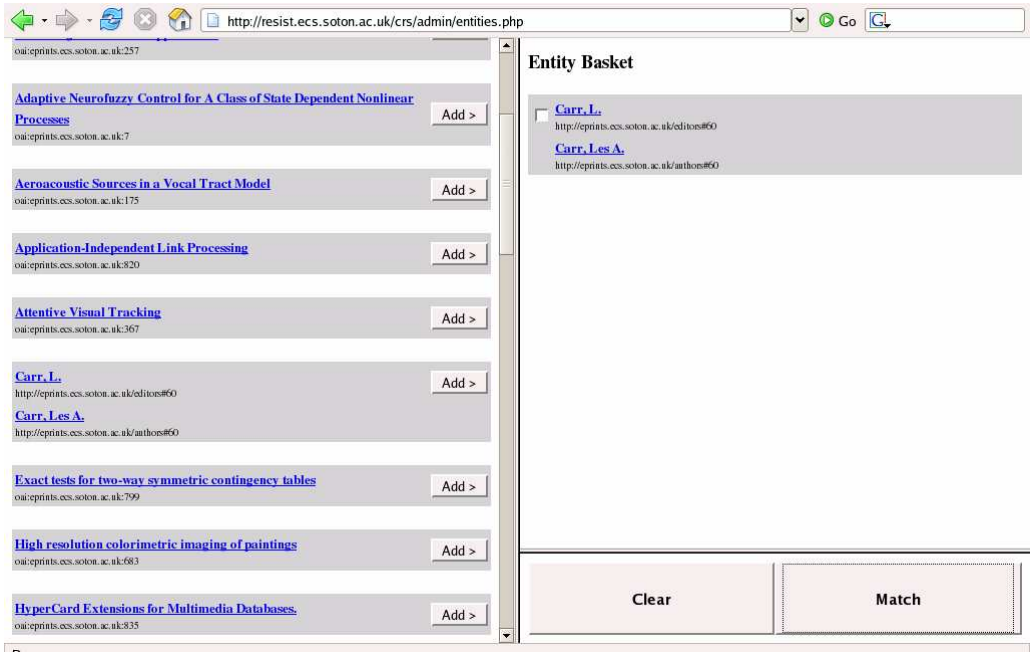


Figure 17. Screenshot of Entity Matcher after Matching.

6 EPrints Extension Design and Implementation

The objectives of the EPrints extensions are to integrate EPrints with the CRS and to exploit the CRS to enhance EPrints as much as possible. There are two extensions. The uploader script extracts entities from the repository and uploads them to the server via the appropriate web services. The plug-in directly enhances the EPrints interface, providing user aids, where possible, to help fill in forms. The plug-in promotes the use of references present in the CRS, preventing the creation of too many new identifiers.

6.1 Uploader

The uploader works by using a small utility supplied with EPrints called 'export_xml'. This dumps the entire contents of a repository onto the standard output. The uploader, which is a PERL script, parses this and detects entities amongst the metadata to upload. The types of entities that the repository administrator wishes to export are specified in a regular expression at the top of the script.

The script works by constructing Entity objects as it parses the data. URIs for the entities are generated either from local identifiers, if present, or if not, by concatenating and hashing all of the Entity's metadata into a single string. Unless a local identifier is detected to already be a URI, identifiers are appended to the CRS namespace by the CRS. Precise details of the process are shown in the activity diagram, Figure 18.

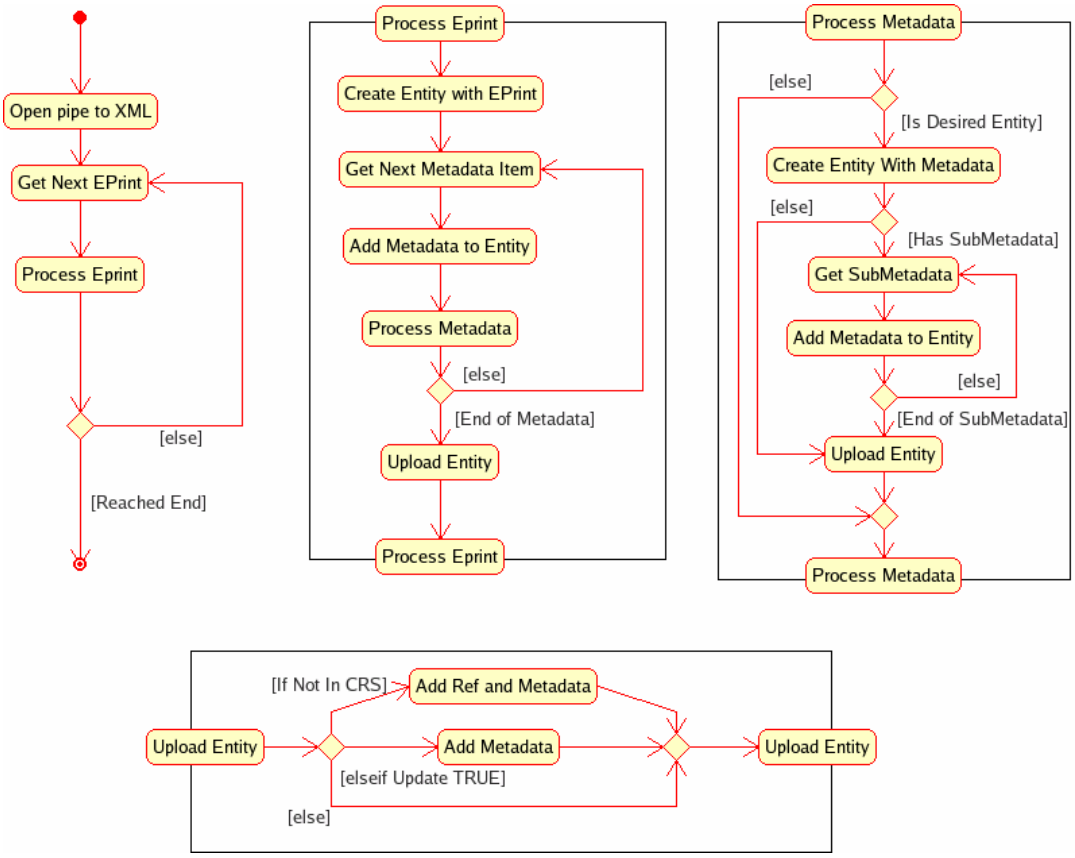


Figure 18. Activity Diagram for the Complete Uploader Process.

6.2 Plug-in

The plug-in for the EPrints interface attaches itself to text fields and attempts to make suggestions as to what the user is wishing to enter. It takes what the user types, performs metadata searches on the CRS and displays the results. In this way, it both helps the user to deposit properly linked articles into the repository and helps the user in performing searches on the repository. For example, if when depositing a document, the user isn't sure of the name of the publisher, or the ID of an author; by entering what they know into the form, the plug-in will suggest possible matching entities and, if selected, will complete the form for the user. There are several key challenges here, notably: only returning results that are relevant to the given field and being able to choose an appropriate ID that reinforces the repository's internal identification scheme.

6.2.1 Install Script

EPrints is highly configurable and abstract. This allows administrators to be in complete control of the look and structure of the repository. It is achieved through heavy use of XML configuration files, templates and PERL scripts to generate pages, as required. As a result of this, very few pages are statically defined in files within EPrints. It is not possible to simply add action listeners manually to every text field. The solution to this is to include a JavaScript program at the top of every page (which is possible as every page shares a common header). When the page is loaded, this program gets every input field present on the page and attaches itself to those which are suitable for enhancement (i.e. are text fields and are visible on the page).

When a suitable field is found, the script records as much information about it as possible. Conveniently, the fields within the EPrints software adhere to a rough naming convention. Input tags that represent hidden or control elements have names starting with an underscore; these can all be ignored by the plug-in. Tags that represent a field for the user to fill in, employ the naming convention shown below.

Entity-Type [_prefix] [_metadata-item]

Example: 'authors_1_surname' or 'authors_1_fname'

This allows the script to determine the field's corresponding entity type, the metadata item that is required and, using the prefix, where multiple fields relate to the same entity. If a field is a general, non type specific search, the entity-type part corresponds to the name of the search.

6.2.2 Search and AJAX

The plug-in is triggered to perform a search by the user entering some text into a field and then either pausing or changing focus to another field. The plug-in opens a connection to the CRS server, using an XMLHTTPRequest AJAX [26] object (Asynchronous Javascript and XML). An HTTP POST request is made to the 'crsbystring' service. It divides the entered text into words, which are used as the search terms. Where multiple fields relate to the same entity, text from each of them is used to generate the terms. The entity type, as discovered by the install script, is used to restrict the search by type. Generic search fields can be specified by a regular expression in the plug-in script, searches on fields matching this expression are not restricted by type.

6.2.3 ID Selection and DOM Injection

AJAX is a set of technologies that allows JavaScript to get and parse information from websites, after the parent page has finished loading. Information acquired can either be treated as text or can be parsed into a DOM hierarchy [27]. When the script sends a POST request to the CRS server, the results are returned in XML and are parsed by the DOM. The object tree is then used to create HTML to display to the user.

The 'crsbystring' script returns the top 5 search results, along with their associated metadata. Each result line represents an entity (a bundle) and all of its references. The plug-in script chooses one references for each entity to recommend to the user. It selects references in the following order, preferring those with identifiers local to the repository:

1. A local reference with a short id (a non URI local ID)
2. A local reference without a short id
3. The canonical reference

Having chosen the appropriate references, the script builds a table in HTML to display to the user. This table shows a line, with a label, for each entity. Local and foreign entities are displayed separately, local first, as generally local references will be preferred over foreign ones. It puts two links on each line; one opens the eninfo.php page that corresponds to the reference and the other selects the reference as the one to use. The plug-in uses DOM injection to insert the HTML into the current page, beneath the field that started the search. Figure 19 shows an example of the plug-in in action.

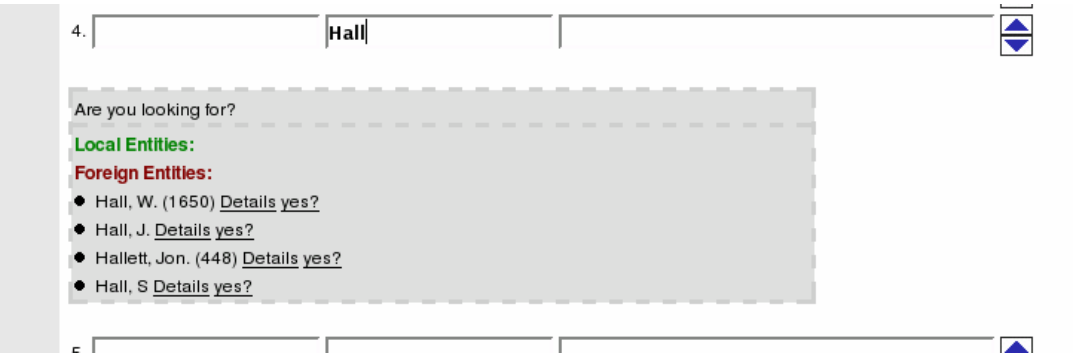


Figure 19 Screenshot of plug-in results table.

6.2.4 Form Completion

When the user chooses an entity, the plug-in fills the form using the metadata supplied with the reference. For each field in the form, the plug-in cycles through the metadata and matches it to an attribute in the data, by constructing and using regular expressions. For ID fields, if available, the plug-in uses a local short id, otherwise it uses the full URI. This means that whenever possible, the plug-in tries to reinforce local ID schemes.

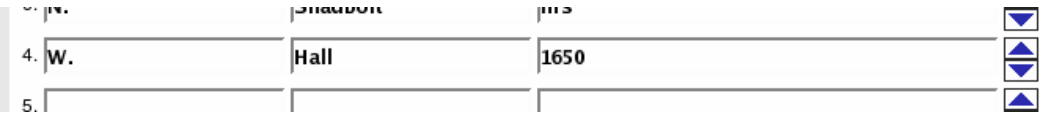


Figure 20. Form having been filled in using the plug-in.

7 Testing

The system was developed using an iterative design and test model. Each time a piece of code was completed, it was immediately tested: the ‘make it and break it’ method. This design and test technique produces a system that evolves and matures over the course of development. The testing at this level is very fine grain, attempting to report the result of every test would be equivalent to trying to report the life history of every line of code in the system. On a higher level, where appropriate, standardised unit testing was performed.

7.1 Unit Regression Testing

The system utilises a number of different languages, none of which are ‘organically’ object orientated. However, despite this, a conscious effort was made to utilise object orientation where available. This provides the benefits of easily readable, reusable and testable code, to otherwise frequently obfuscated languages.

The CRS runs on a fully OO backend, as such and due to the importance of this subsystem, a durable set of test harnesses was developed to ensure the consistent integrity of the CRS’ essential functions. The unit testing system used was ‘Simple Test’ [28]. This package was chosen for its flexibility and ease of use.

For each of the three central classes in the CRS backend, a test harness was constructed to perform a series of tests, to ensure the essential functions continued to work as prescribed. The tests are run from a single web page, where test results and errors, if any, are displayed. This allows the CRS to be automatically tested every time any changes or bug fixes are implemented. See Appendix D for a screenshot of the unit test results page.

Unit testing was performed on the other subsystems. For the EPrints plug-in, the script was supplied with a sample XML result set. The test was shown to be successful as the plug-in was able to display the results as a table and was able to fill the forms using the metadata. The web services were tested by comparing their behaviour, given specific parameters, to what they were expected to do. Having already tested the CRS backend, this was a simple and straightforward task.

7.2 Deployment / Scalability Testing

During development of the project, a considerable amount of third party interest was generated. One party was the ReSIST Project [29]; they wished to make use of a CRS server, combined with the data from the ECS EPrint Server (eprints.ecs.soton.ac.uk). To this end, they donated the use of their server for the deployment of a CRS. From this arrangement, the project benefited from having a live and substantial server to perform a test deployment on.

In order for the system to be deployed on a third party machine, the code had to be refactored into a format that was highly portable and configurable. This was achieved by improving the system’s directory structure, ensuring all the required files belonged to a single hierarchy, and by reinforcing the use of system wide constants. This exercise proved to be a successful test of the systems readiness for live deployment. The only

problems encountered were compatibility issues due to differences between the older version of PHP, on the ReSIST server, and the latest version, on the project test bed. To overcome this, the syntax used was converted to the older standard.

Having successfully deployed the server, scalability tests were performed to show that the CRS is capable of handling the amount of data present in a sizeable live deployment. Two subsystems were tested: The EPrints uploader and the crsbystring service. The uploader was tested by timing how long it took to upload an increasing number of EPrints. The web service was tested by comparing the response time for the service with different quantities of data in the server. The crsbystring service was chosen as it performs the most complex queries. It would therefore be the first subsystem to show a drop in performance and is an ideal candidate for testing. See below for the results.

Number of EPrints	Estimated Number of Unique Entities	Cumulative Upload Time (hh:mm:ss)	CRS response time
10	50	00:00:11	< 1 sec
100	500	00:01:23	< 1 sec
200	1000	00:02:32	< 1 sec
500	2000	00:06:12	< 1 sec
1000	4000	00:13:23	< 1 sec
2000	8000	00:23:35	< 1 sec
5000	20000	01:01:17	< 1 sec
10000	40000	01:49:35	< 1 sec

Figure 21. Scalability Test Results.

The uploader was shown to scale in roughly linear fashion. It took almost two hours to upload the full, 10,000 EPrint, repository. Though as the uploading process would probably be performed only once a week and considering that this was a very sizeable test set, two hours is an entirely acceptable time. The performance of executing a search on the repository did not noticeably degrade with the increasing number of entities: a very favourable result, as the performance of the query engine is crucial to the performance of any plug-in, or software utilising the CRS. Less than one second is an acceptable time for the user to have to wait for the system to display suggestions. In many cases, the user will not be expecting to receive help and so will not be aware of the lag at all.

7.3 Informal UAT / Sponsor Feedback

The overall functionality and appropriateness of the system in fulfilling the requirements is crucial to the success and future uptake of the system. However, it is difficult to quantitatively test such abstract qualities. Fortunately, the system has been demonstrated to a wide range of interested people. The broad range of feedback received reflects the success of the project.

The system has been demonstrated to members of Southampton ECS, the AKT IRC, who developed the original coreference framework, the ReSIST Project and the EPrints

development team. Feedback was very positive, as was shown by the different party's reactions.

As discussed, the ReSIST Project wished to make immediate use of the system; they came to the arrangement of allowing their server to be used for testing, in exchange for the deployment of a CRS.

The AKT Project had actually previously applied for several thousand pounds of funding for a similar system. They were very impressed with its efficiency and drew attention to the significance and implications of such a system, strongly encouraging its integration with a future version of EPrints. See 51Appendix E for a report of the feedback from Hugh Glaser, a member of the AKT and ReSIST projects.

The EPrints development team, who were contacted in connection with cloning the ECS repository for the use of the ReSIST project, were very interested. They remarked that the system provided effective solutions to several key problems with future extensions to EPrints.

Generally feedback was very positive. Praise was particularly given regarding the sophistication and integration of the plug-ins with the EPrints repository. Those in the semantic web field identified the usefulness of the system in a wider community; whilst people from other areas remarked upon the significant benefits provided for EPrint users and administrators, by the plug-in.

Some constructive criticism was received. Minor bugs were identified within the administration interface, which came to light by allowing other users to utilise the interface with the browser of their choice. It also became apparent that the plug-in's initial restriction of displaying only 4 entities at a time was slightly insufficient and so was increased to 5.

8 Evaluation

8.1 Requirements

The feedback and test results strongly indicate that the project is both successful and suitable for combating the problems it was designed to address. The CRS fulfils its requirements of being able to track equivalences between references, be highly compatible and provide useful and efficient interfaces. Through rigorous testing the system has been proven to be highly responsive, even when dealing with very large amounts of data. It has also been shown to be ready for live deployment and use. The table below shows a summary of the system's requirements and achievements.

Requirement	Achievement
Enable Cross Referencing	The system records mappings between identifiers for any entity type, allowing cross referencing between any system, not just EPrints
Enhance EPrints	The plug-in provides auto complete functionality for all of EPrint's forms. It suggests completions for fields regarding any type of information.
Reinforce Existing Identifiers	The plug-in uses the CRS to find existing local identifiers, which it uses in favour of others, keeping the repositories identifiers consistent.
Be Open	The system employs very lightweight web services that can be integrated easily into any system that can make HTTP requests.
Be Flexible	The 3store backend allows any type of entity or metadata to be seamlessly added to the system.
Be Scalable	The backend scales in a linear fashion. It has been successfully tested with up to, approximately, 40000 entities.
Be Deployable	The system is portable, requiring nothing more than PHP and MySQL. It is readily deployable and highly compatible, as testing has shown.
Be Efficient	The system uses the minimal number of triples to record each equivalence and reference. It achieves this through the use of advanced coreference techniques

Figure 22. Table of comparing the CRS' requirements and achievements.

Feedback regarding the EPrint plug-in was equally supportive. Comments indicated that the utilisation of the CRS to aid the user in interfacing with EPrints was extremely useful; specifically, the effort required in depositing articles into the repository was significantly reduced.

8.2 Suitability and Competition

In terms of competition, there is little to compare the CRS to. For allowing interoperation between repositories, the only other systems available are those bespoke solutions, built into specific applications. These do little in the way of promoting the use and usability of Institutional Repositories, as semantic data sources. The CRS is open to all to use and contribute to. It facilitates easier application development, to the point that it could potentially support an open community of academics, aimed at providing software for gaining greater returns from the members' self-archived, academic output.

For adding enhanced interface features to the EPrints software, the only similar system is the ACIS project. Section 2.4.2 outlines the main shortcomings of the ACIS system. See the below table for a comparison of the project to alternative solutions.

Alternative	Comparison
CRS	
ACIS	Only aids registered users. The system is aimed at linking articles back to the ACIS and does nothing to improve internal consistency. It provides no interoperability with other systems, as the CRS does.
Naming Authority	Would be hard to implement, poorly scalable and would restrict open growth. The CRS allows repository owners the freedom of choosing their own identifiers.
Automatic Mapping	Very unreliable. Would only work with a significant amount of metadata available. The CRS creates an open forum for mapping where anyone can contribute results or corrections.
Do Without	Metadata from different repositories would remain incompatible, making it virtually impossible to provide useful services based on data from IRs.
EPrints Plug-in	
EPrints Internal Auto complete	Whilst this would reinforce the internal consistency, it would be unable to provide linking with external entities. The CRS both promotes the use of internal identifiers and provides identifiers for foreign entities.
Do Without	Identifiers within EPrints would continue to be inconsistent. Data from within repositories would be hard to correlate, let alone data between repositories.

Figure 23. Table comparing the CRS against alternative solutions.

8.3 Reflections

The CRS provides functionality allowing institutional repositories and potentially many other semantic data providers to more easily interoperate and share data. However, its success is dependant not only on its functionality and performance, but on its uptake by the community as well. If the CRS is not utilised, then it is unable to provide useful services. This project has developed the system, but it cannot ensure its use.

To ensure the CRS enjoys true success, further investigations should be carried out to find how best to build a community of users and systems to utilise it. One way to achieve this might be to deploy relatively small servers at first, providing functionality for domains of a limited size. Once established, these could then be merged or expanded to enlarge their user bases.

Adoption would also be better facilitated if the client software (the EPrints extensions) were made widely available, reducing any disincentives to employing a CRS. Perhaps the plug-ins could be provided optionally, or even as standard, with a future EPrints release. This has already been proposed (see Appendix E).

9 Future Work

Whilst the CRS developed by this project is, in itself, a fully developed system, ready and able for deployment and use; there are inevitably further avenues of research that might provide enhanced functionality. This section outlines some proposals for extensions that have arisen during development.

9.1 Further Extensions to EPrints

At current the EPrint plug-in displays, for each entity, the label and the ID. For more information, the user may visit the entity info page. With a CRS server populated by very many different entities, there is a higher chance of very similar entities appearing, that are not equivalent. This would provide a minor inconvenience for a user of the system as it is, they would have to refer to the info page to choose between the entities. A more sophisticated solution to this would be to employ something similar to what is used by the Internet Movie Database (IMDB)[1]. There, when searching for a film or actor and there are multiple matches of the same name, the results are listed along with a single item of most significant metadata. Searching for 'Robin Williams', for instance, returns 'Robin Williams (Actor, Good Will Hunting)' and 'Robin Williams (Miscellaneous Crew)'. This allows the user to very easily make a decision, without being overloaded with data. Within EPrints, this might be achieved by using the most recent piece of work for an academic, or for a document, the most noteworthy author.

If the mapping process is performed entirely manually, it would prove a very time consuming and arduous task for a single administrator. The job might be better achieved if the matching interface was integrated into the EPrint registered user pages. Each user could take responsibility for matching entities relating to their deposits. This would be a step closer to the ACIS system described in section 2.4.2. Though, rather than having a user's effort in mapping go solely towards CV generation, they would leverage many advantages and useful applications. It would also not require registration with a separate interface to EPrints, as ACIS does.

Institutions that decide to gain greater returns, by using a CRS system with their existing repository, do not enjoy improved internal consistency of articles that are already present. This is because the EPrint plug-in does not attempt to correct metadata, once it has been entered into the repository. An interface, or script, could be designed that would highlight possible discrepancies within the metadata and suggest alternative values.

It is planned that in the near future EPrints will have better support for third party extensions. This will most likely be in the form of a modular system. Plug-ins, written implementing the appropriate interfaces, would be able to link automatically into the EPrints system and be supplied with required data. The implications of this would be greater efficiency, ease of development for extensions and greater interest from developers in the EPrints software. If, as and when this new version of EPrints is

[1] The Internet Movie Database (IMDB) and all related content and technologies are Copyright © 1990-2006 Internet Movie Database Inc.

released, the CRS plug-ins were rewritten to take full advantage of the new functionality, the system might enjoy faster and smoother integration.

An obvious extension to the project would be to provide plug-ins for repositories other than EPrints. EPrints was chosen due the interest and proximity of the EPrints team and the availability of support. A plug-in for DSpace could, perhaps, be developed by someone with similar resources available.

9.2 Extensions to the CRS Server

The CRS is designed to be integrated with external mapping processes, but does not by default provide any automated matching facilities. It might add to the system if some form of 'in house' mapping system were made available. Perhaps not to perform the task automatically, as this has inherent risks, but to provide suggestions and to highlight possible duplicates in the administration interface. It would then be left to the user, or administrator to accept or ignore the suggestions.

Whilst a CRS is designed to provide functionality only to the community it was set up for, there are conceivable circumstances in which one might wish to link two servers together. If, say, two areas of computer science had their own CRS servers and at some point, after they were set up, the work of these two areas started to merge; one may well wish to be able to use information from both servers. The current recommended solution for this would be to merge the two CRS servers into one. However, if the arrangement was temporary, merging would not be ideal. A way round this would be the ability to link CRS servers together. This could be achieved by a simple linking, whereby a search in one CRS also searched the other.

Alternatively, a more sophisticated solution: when a new reference is added to a local CRS, it could be searched for in a list of affiliated servers. If the reference is found elsewhere, some form of remote link could be added to the local CRS, pointing to the location of the other server. This would allow the use of the foreign bundle locally, whilst leaving control and ownership of the bundle in the hands of the foreign server. The foreign bundle would be returned whenever a search matched the reference that was added to the local CRS. This solution would be much more efficient than searching laboriously through every server, for every search. Foreign servers would only be searched on the addition of the reference, thereafter the foreign bundle can be addressed directly via the remote link.

The web services used by the CRS are deliberately lightweight and simplistic. This keeps the weight of implementation, and therefore the server load, down. However, a number of new semantic web applications are orientated around web service standards, such as SOAP [30]. To facilitate the use of the system by these applications, an alternative set of web services could be developed that provide a more standard compliant interface. Alternatively, a standardised, directory like, system could be supplied. By utilising a system such as LDAP [31], the contents of the CRS could be accessed by anyone, whether they were familiar with semantic web technologies or not. Such extensions, while useful, were deemed to be peripheral to the central goals of the project, and therefore fell outside its scope.

10 Final Conclusions

Current applications attempting to gain added value from EPrint repositories have to overcome significant hurdles in order to produce coherent results. These obstacles provide disincentives for developers to provide otherwise extremely useful and timely applications.

This project helps to overcome these barriers. It achieves this by providing and utilising a consistent reference service, which maps between the different identifiers both within and between different repositories. It allows references to be mapped both manually and by new or existing mapping processes. Using these mappings, applications are able to cross-reference data from multiple sources, in order to provide useful and interesting services.

The CRS utilises semantic web techniques to efficiently store metadata and provide both XML and RDF output, allowing for maximum compatibility with 3rd party applications. The bundle structure used stores references and mappings in a retrievable and scalable manner, whilst the unique division of data into specific, identifiable, graphs allows equivalences to be easily updated and manipulated. Carefully optimised SPARQL queries ensured that data retrieval is performed in the most efficient time.

The EPrints plug-in aids users in completing forms within repositories. This not only helps to make the metadata more consistent from the outset, but also makes interacting with EPrints significantly easier. The plug-in uses dynamic HTML and JavaScript (AJAX techniques) to obtain data, make suggestions and fill forms without ever having to reload the page. The plug-in intricately installs itself only onto suitable fields and is able to restrict searching to the specific types of entities relevant to each form.

Through rigorous testing and the large volume of favourable feedback that the CRS has received; the system has been demonstrated to be readily deployable, scalable and highly usable. It has even been deployed on a live server, for the use of researchers in the ReSIST project.

With continued interest and uptake, the CRS represents an original and efficient method for tackling the problem of referential inconsistencies, not only for institutional repositories but within the semantic web at large as well.

References

[1] Great Britain, House of Commons, Science and Technology Committee, Scientific Publications: Free for all? Tenth Report of Session 2003-04 HC 399- I & II, 2004, London, The Stationery Office Limited ISBN 0215018419 for HC 399-I, 0215018419 for HC 399-II web reference: http://www.publications.parliament.uk/pa/cm200304/cmselect/cmsctech/399/399.pdf

[2] C. Gutteridge. GNU EPrints 2 Overview. In Proceedings of 11th Panhellenic Academic Libraries Conference, Greece, 2002.

[3] M. Smith et al. DSpace: An Open Source Dynamic Digital Repository, D-Lib Magazine, January 2003.

[4] RAE 2001 Results. http://www.hero.ac.uk/rae/Results/, 2001.

[5] H. Glaser. Consistent Reference Service. Call for Projects in Digital Repositories. JISC Circular 03/05

[6] C. A. Lynch. Metadata Harvesting and the Open Archives Initiative. ARL Bimonthly Report 217, August 2001.

[7] C. Lagoze, H. Van de Sompel, N. Nelson and S. Warner. (editors), The Open Archives Initiative Protocol for Metadata Harvesting v2.0, 2002.

[8] Dublin Core. Dublin Core Metadata. In http://purl.org/metadata/dublin_core, 1997.

[9] Brody, T. Citebase Search: Autonomous Citation Database for e-Print Archives. http://eprints.ecs.soton.ac.uk/10206/, 2003.

[10] O. Lassila and R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. W3C recommendation, 1999.

[11] D. McGuinnes, F. van Hermelen. OWL Web Ontology Language Overview. W3C recommendation, W3C, Feb 2004.

[12] T. Berners-Lee, Hendler J., and O. Lassila. The semantic web. *Scientific American*, May 2001.

[13] H. Alani, S. Dasmahapatra, N. Gibbins, H. Glaser, S. Harris, Y. Kalfoglou, K. O'Hara, and N. Shadbolt. Managing Reference: Ensuring Referential Integrity of Ontologies for the Semantic Web. In 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW02), pages 317-334, Siguenza, Spain, 2002.

[14] R. Reiter. Equality and domain closure in first order databases. Journal of the Association of Computing Machinery, 10(4):334-350, 2001.

[15] H. Alani, K. O'Hara, and N. Shadbolt. ONTOCOPI: Methods and tools for identifying communities of practice. In Proceedings of the 2002 IFIP World Computer Congress, Montreal, Canada, August 2002.

[16] E. Wenger. Communities of Practice: The Key to Knowledge Strategy. Cambridge University Press, 1998.

[17] X. Dong, A. Halevy, and J. Madhavan. Reference Reconciliation in Complex Information Spaces. Technical Report 2005-03-04, Univ. of Washington, 2005.

[18] S. Harris and N. Gibbins. 3store: Efficient bulk RDF storage. In Proceedings of the 1st International Workshop on Practical and Scalable Semantic Systems (PSSS'03), Sanibel Island, Florida, pages 1-15, 2003.

[19] T. Lewy, H. Glaser and N. Shadbolt. A Framework for Reference Management in the Semantic Web. Submitted to World Wide Web Conference 2006.

[20] Academic Contributor Information System Project. http://acis.openlib.org/, 2006.

[21] T. Krichel and I. Kurmanov. ACIS Stage Three Plan, http://acis.openlib.org/stage3/, 2005.

[22] T. Krichel and I. Kurmanov, ACIS project: phase 1 requirements, http://acis.openlib.org/documents/kathmandu.html, 2003.

[23] SPARQL Query Language for RDF. W3C Working Draft. http://www.w3.org/TR/rdf-sparql-query/, 2005.

[24] RDF Vocabulary Description Language 1.0: RDF Schema W3C Recommendation http://www.w3.org/TR/rdf-schema/, 2004.

[25] D. Westphal and C. Bizer. Introduction to RAP. RAP - RDF API for PHP Documentation, http://www.wiwiss.fu-berlin.de/suhl/bizer/rdfapi/tutorial/introductionToRAP.htm, 2004.

[26] J. J. Garrett. “Ajax: A New Approach to Web Applications”. http://www.adaptivepath.com/publications/essays/archives/000385.php, 2006.

[27] Philippe Le Hégaret, Document Object Model. http://www.w3.org/DOM/, 2005.

[28] Simple Test for PHP. Marcus Baker. http://www.lastcraft.com/simple_test.php, 2006.

[29] ReSIST: Resilience for Survivability in IST. http://www2.laas.fr/RESIST/index.html, 2006.

[30] SOAP Version 1.2 Part 1: Messaging Framework W3C Recommendation http://www.w3.org/TR/soap12-part1/, 2003.

[31] Yeong, W., Howes, T., and S. Kille, "*Lightweight Directory Access Protocol*", RFC 1777, March 1995.

Appendix A. Bundle structure

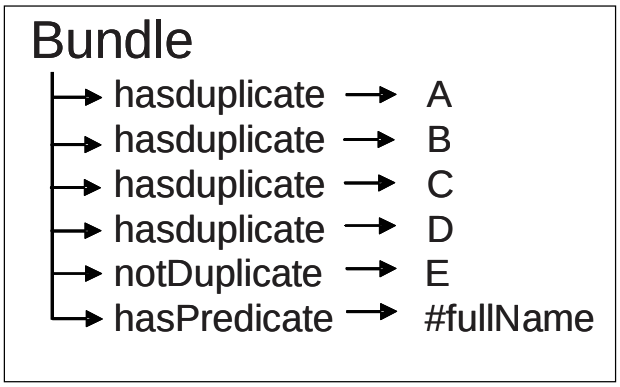


Figure 24. Visualisation of a bundle

```
<?xml version="1.0" encoding="UTF-8" ?>

<rdf:RDF

   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:ns1="http://www.aktors.org/ontology/coref#">

<ns1:Bundle rdf:about="http://www.aktors.org/ontology/coref#bundle-
26cd8e445b1254ffd8663d13c588e394">

   <ns1:duplicate
rdf:resource="http://nlp.shef.ac.uk/#ARM_AUTHOR_Hugh__Glaser"/>

   <ns1:duplicate
rdf:resource="file:/usr/local/share/akt/ResearchMap/akt-map-
onto.rdf#Knowledge-Services"/>

   <ns1:duplicate
rdf:resource="http://www.ecs.soton.ac.uk/info/#person-00021"/>

   <ns1:duplicate
rdf:resource="file:/usr/local/share/akt/ResearchMap/akt-map-
owl.rdf#DOME"/>

   <ns1:duplicate
rdf:resource="http://www.ecs.soton.ac.uk/info/#person-00021"/>

   <ns1:hasPredicate
rdf:resource="http://www.aktors.org/ontology/portal#full-name"/>

</ns1:Bundle>

<rdf:Description rdf:about="http://www.ecs.soton.ac.uk/info/#person-
00021">
   <ns1:isCanon
rdf:resource="http://www.aktors.org/ontology/coref#bundle-
26cd8e445b1254ffd8663d13c588e394"/>
</rdf:Description>

</rdf:RDF>
```

Figure 25. Example Bundle RDF.

Appendix B. Schedule

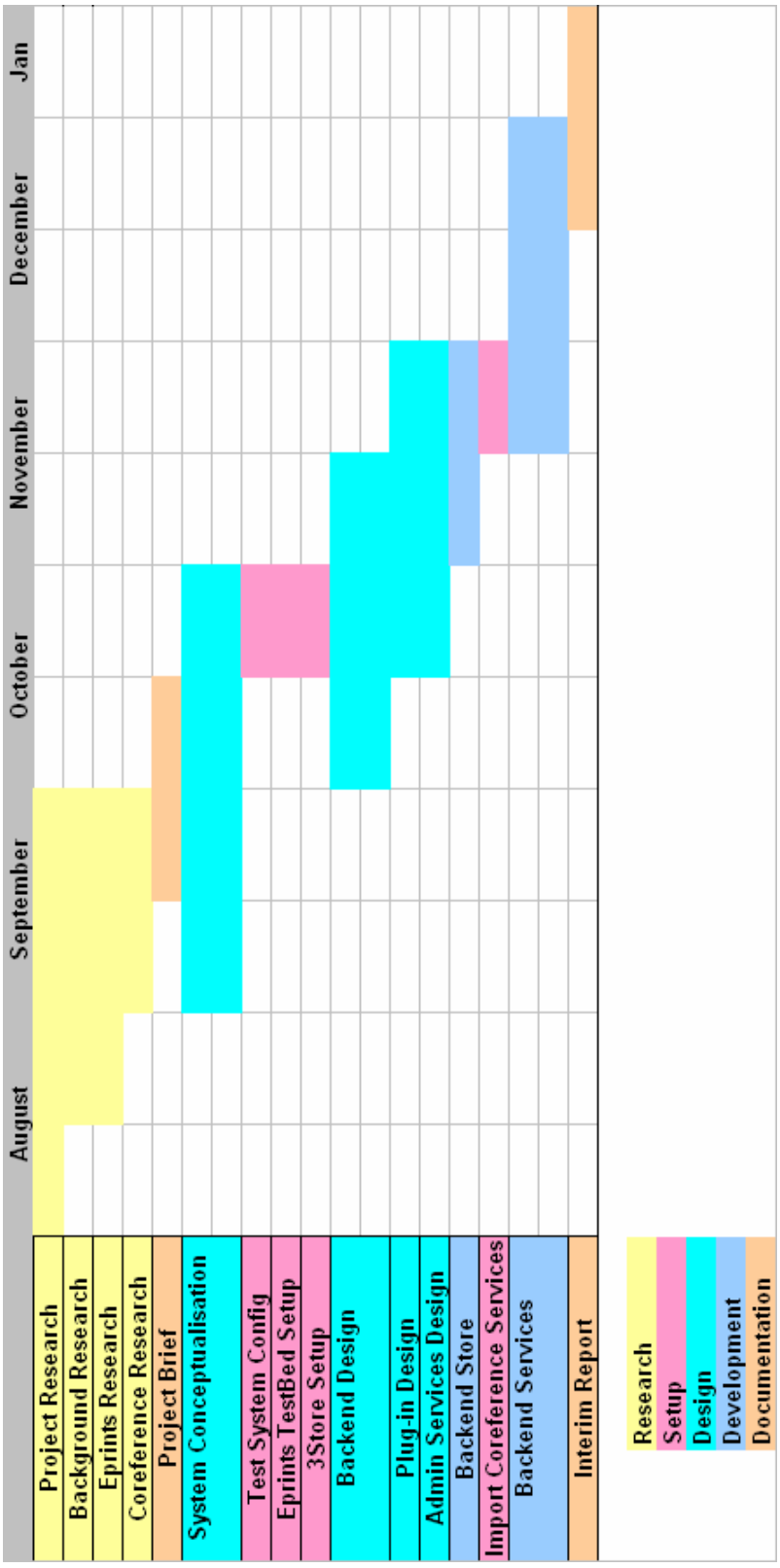


Figure 26. Gantt Chart Schedule of Project Development Stage 1.

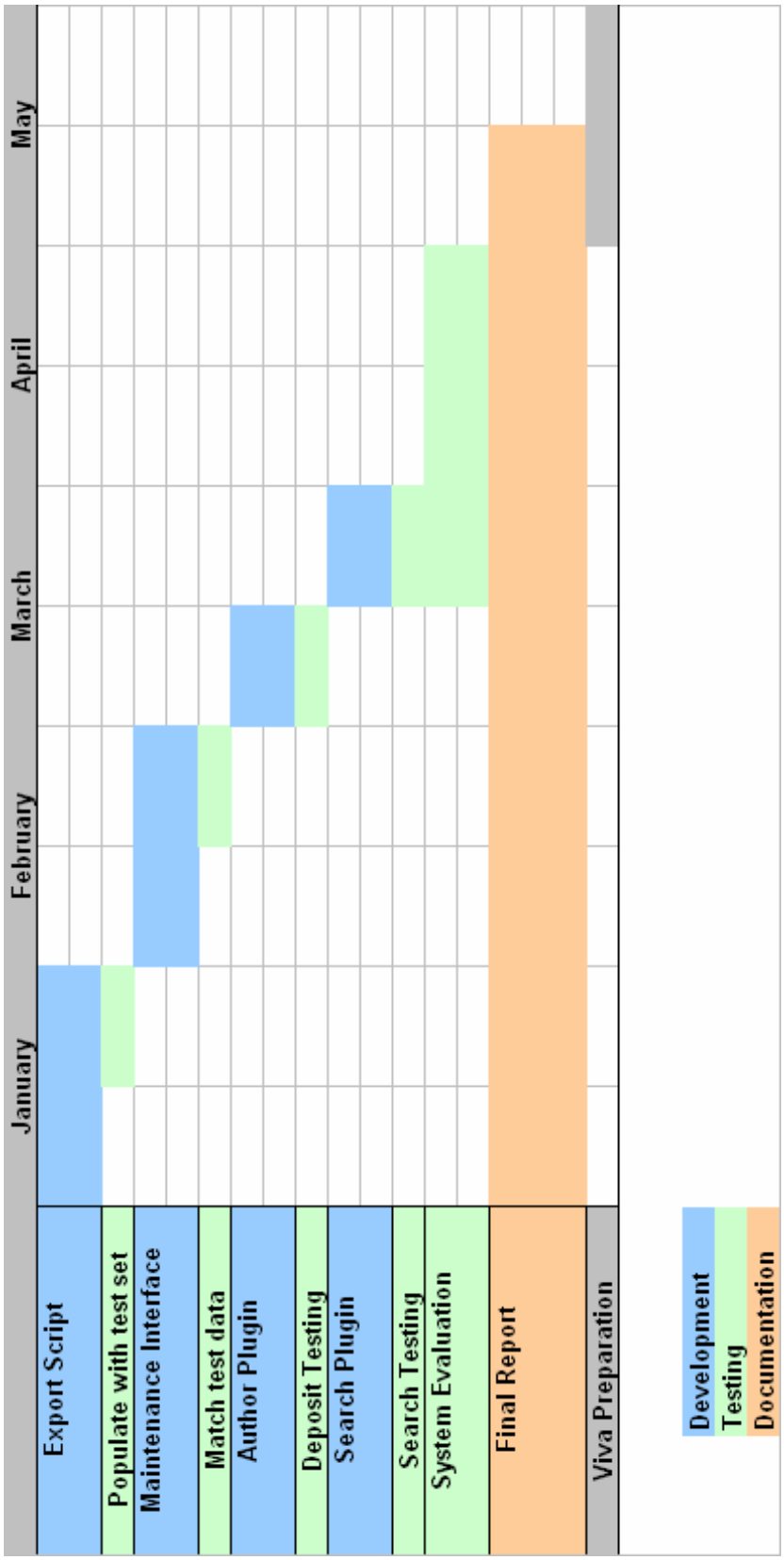


Figure 27. Gantt Chart Schedule of Project Development Stage 2.

Appendix C. Class method and attribute details

Bundle

The Bundle class models the behaviour of a Bundle structure. It provides methods for performing essential operations on bundles. The RAP RDF API for PHP is used to generate RDF.

Private Properties:

date: int
Date integer, for recording when the bundle was last updated. Is an integer corresponding to a ddmmyyy date format.
canon: string
URI of the canonical reference. Is set automatically using a lexicographical ordering.
references: array
Array of URIs for the references contained by the bundle.
name: string
Name of the bundle. Automatically generated as references are added. It corresponds to the string 'bundle-' concatenated to the MD5 hash of the canonical reference's URI.

Public Methods:

getModel();
Returns the RAP model object corresponding to the bundle's RDF.
string getURI();
Returns the URI of the bundle.
string getName();
Returns the name of the bundle.
setCanon();
Forces the canonical reference to be recalculated.
setDate(int date);
Sets the date attribute to the passed integer.
int getDate();
Returns the date attribute.
addReference(string reference);
Adds the passed URI to the array of references. Also recalculates the canonical reference and bundle name in case the new reference replaces the canon.
removeReference(string reference);

Removes the reference corresponding to the passed URI from the bundle.
merge(Bundle bundle);
Merges the current bundle with that passed. Returns the new combined bundle.
string getRDF();
Returns an RDF XML document, as a string, corresponding to the bundle.
string getXML();
Returns a plain XML document, as a string, corresponding to the bundle.
int getNumReferences();
Returns an integer corresponding to the number of references present in the bundle.
array getReferences();
Returns the array of reference URIs.
string getCanon();
Returns a string corresponding to the URI of the canonical reference. If the canon has not yet been set, it is calculated before being returned.
makeNameUnique();
Forces the name of the bundle to be recalculated from the canonical URI.

BundleSet

The BundleSet class provides operations for correctly handling multiple bundles simultaneously. The RAP RDF API for PHP is used to generate RDF.

Private Properties:

bundles: array
Array of all the Bundle objects in the set.

Public Methods:

char getNumBundles();
Returns an integer corresponding to the number of Bundle Objects in the set.
array getBundles();
Returns the array of Bundle objects.
addBundle(Bundle bundle);
Adds a passed bundle object to the set.
removeBundle(string bundlename);
Removes the Bundle in the set that corresponds to the passed bundle name.

string getXML();
Returns a single plain XML document, as a string, corresponding to the all the bundles in the set.
string getRDF();
Returns a single RDF XML document, as a string, corresponding to the all the bundles in the set.
getModel();
Returns the RAP model object corresponding to the bundles RDF.

BundleStore

The BundleStore class models the behaviour of a bundle enabled triplestore. It interacts with a 3store knowledgebase via a SPARQL web service and the 3store import script. It provides methods for inserting, removing and retrieving bundles from the database. The RAP RDF API for PHP is used to generate RDF.

Private Properties:

sparql: string
URI of the SPARQL service to be used by the SPARQLer object.
import: string
Location of the 3store 'ts-import' script being used to import RDF.

Public Methods:

Bundle getBundleByRef(string RefURI);
Method for returning a bundle from the store, given any one of the references that it contains. Returns a Bundle object if found, 0 if not.
Bundle getBundleByName(string BundleName);
Method for returning a bundle from the store, given its name. Returns a Bundle object if found, 0 if not.
Bundle getBundleByURI(string BundleURI);
Method for returning a bundle from the store, given its URI. Returns a Bundle object if found, 0 if not.
BundleSet getBundlesByMetadata(array arrayofstrings, int limit, string origin, string type, int fromdate);
Method for searching for bundles in the store by metadata. The method takes an array of strings that are used as keywords for searching. The search can optionally be restricted to: bundles containing references of a specific origin; bundles newer than a certain date; or bundles containing at least one reference of a type equivalent to a specified type. The maximum number of bundles returned can also be specified. Returns a BundleSet object if found, 0 if not.

insertBundle(Bundle bundle);
Method for inserting a new Bundle into the knowledge base.
deleteBundle(string bundlename);
Removes the bundle from the store that corresponds to the passed bundle name.
string getXMLWithMetaData(Bundle bundleorset);
Returns XML for the passed Bundle(s), with all corresponding metadata from the knowledge base added.
string getRDFWithMetaData(Bundle bundleorset);
Returns RDF for the passed Bundle(s), with all corresponding metadata from the knowledge base added.
newType(string newtype);
Adds a new type to the bundle store. Adds the new type to its own bundle and asserts it to be a rdfs:Class.
setMetadata(string ref, array metadata);
Sets or updates the metadata for the given reference. Metadata is supplied as an associative array where the keys correspond to the predicates and the values to the objects. Any predicates that are not URIs will be appended to the CRS namespace being used.

SPARQLer

The SPARQLer class performs SPARQL queries upon a 3store and returns the results as associative arrays. It uses a SAX parser to read the results from the XML output by 3store.

Private Properties:

db: string
String corresponding to the 3store database that is being used.
server: string
String corresponding to the URI of the SPARQL web service that is being used for querying.

Public Methods:

array performQuery(string query);
Performs the passed SPARQL query and returns the results as an associative array. The keys of the array represent the variable names and the values the results. The keys [variablename]_type is used to return the type (literal of URI) of the variable value.

Appendix D. Unit Testing Screenshot

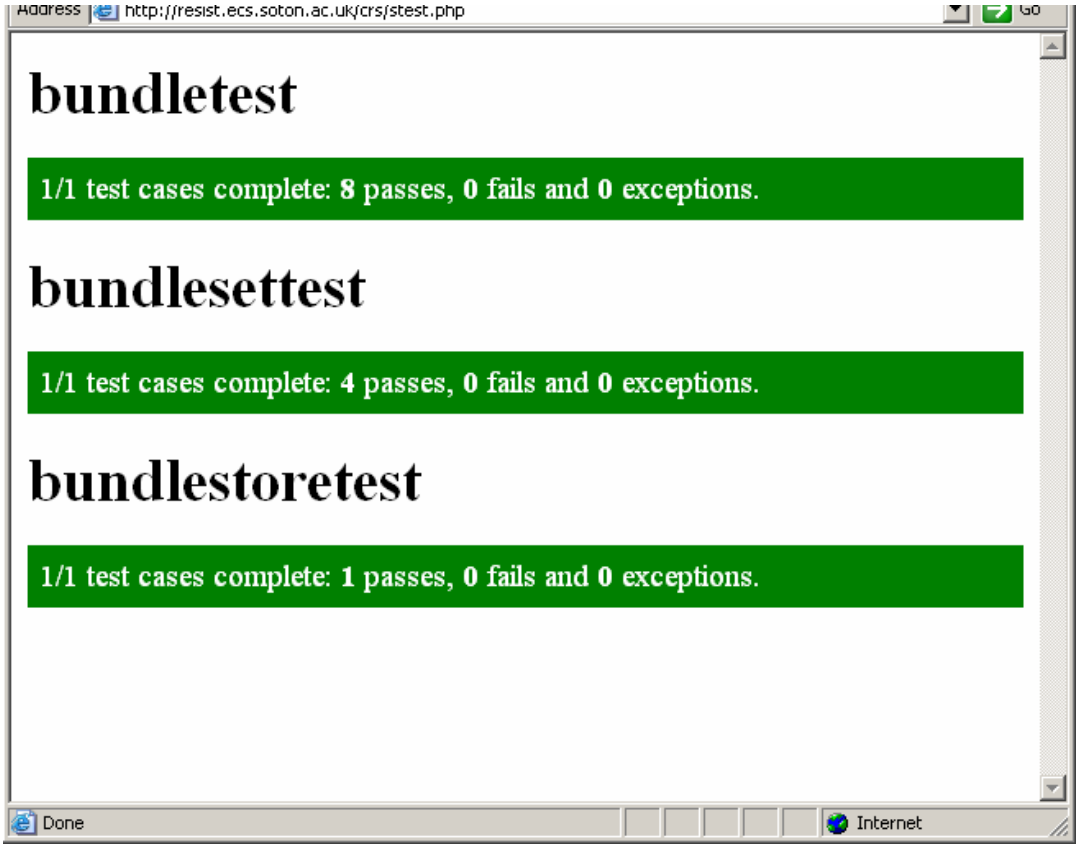


Figure 28. Screenshot of Unit Testing Results Page.

Appendix E. Report on Hugh Glaser's Remarks

University of Southampton

School of Electronics and Computer Science
IAM Research Group
Eric Cooke, Senior Tutor

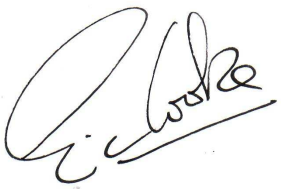
University of Southampton
Highfield
Southampton
SO17 1BJ United Kingdom

T +44 (0)23 8059 3271
F +44 (0)23 8059 2865
E ecc@ecs.soton.ac.uk
www.ecs.soton.ac.uk

9th May 2006

Hugh Glaser's opinion on Timothy Lewy's Project

Hugh Glaser and Les Carr submitted a grant proposal for £90K to satisfy the specification of this project. In the event, the work has been done by Timothy. Hugh is very satisfied by the project: it proves the value of his concept of general plug-ins for ePrints and it improves the functionality of ePrints by implementing a Consistent Reference Service (CRS). The project is currently successfully running on a clone of the ECS ePrints repository and it will be released with future releases of ePrints.



Eric Cooke