Hafslund Sesam – An Archive on Semantics

Lars Marius Garshol and Axel Borge

Bouvet ASA, Oslo, Norway
{larsga,axel.borge}@bouvet.no

Abstract. Sesam is an archive system developed for Hafslund, a Norwegian energy company. It achieves the often-sought but rarely-achieved goal of automatically enriching metadata by using semantic technologies to extract and integrate business data from business applications. The extracted data is also indexed with a search engine together with the archived documents, allowing true enterprise search.

1 Introduction

Every enterprise has a number of different IT systems, each of which maintains an incomplete picture of the enterprise. The full picture is nowhere to be found, because information is not connected across the different systems. Solving this is non-trivial, as traditional systems can only store data which fits their schema, and a single system for the entire enterprise is unrealistic.

We have developed a system called Sesam for Norwegian energy company Hafslund, which collects information from different IT systems and integrates it into a meaningful whole. This allows users to search and browse data across system borders. The system avoids the schema problem by using RDF to store the integrated data.

Sesam is actually Hafslund's internal document archive, but an archive built in an unusual way. Documents are tagged with URIs from the triple store, and these URIs connect the document metadata with enterprise data extracted from backend systems. Having the enterprise data available also allows metadata to be automatically enriched by traversing the data in the triple store.

The system thus improves metadata quality while at the same time reducing the need for manual metadata input by users. In addition, it is used by customer service representatives to find information relevant to callers.

An overview of the system architecture is shown in figure 1 on the facing page.

1.1 User Interface

The user interface to the system is an application built on a search engine, which has indexed both documents and the structured RDF data. The application presents a faceted search interface with entity pages (pages that show all data about one entity), and the ability to navigate from one entity to related entities.

P. Cimiano et al. (Eds.): ESWC 2013, LNCS 7882, pp. 578–592, 2013.

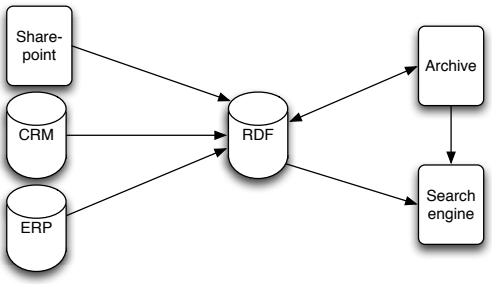

Fig. 1. System architecture

The interface also provides type-ahead functionality to help users understand what they can search for.

In the user interface system users can navigate from a customer in the CRM system to the same customer in the ERP system, and from the ERP customer to connected equipment in the ERP system, and so on.

The user interface is deliberately kept generic, the display logic for RDF data being a direct translation from the structure of the RDF data. Thus new properties and classes can be added to the RDF data and be displayed without modifying the user interface.

1.2 Collecting Information

All source systems are integrated in the same way: a wrapper is added to expose an SDShare server interface. SDShare is a specification for synchronizing RDF data using Atom feeds [SDShare]. Once a source system exposes a set of SDShare feeds the integration is complete, as an SDShare client can then pull the data into the triple store.

The SDShare client is a generic implementation of the SDShare protocol, which periodically polls each data source for new data, and automatically transferring any new data to the triple store, keeping the triple store in sync with sources. At the moment the client polls most sources every 5 minutes, which is more than sufficient for an archive system. Some sources are polled more often, and some as rarely as once an hour.

Data from each source system is kept in a separate graph in the triple store, allowing the source of each statement to be tracked. This also provides a partitioning of the data that is useful for maintenance purposes.

1.3 Archiving

Sesam exposes a web service interface for archiving based on the CMIS standard [CMIS], to allow applications to add support for archiving directly from the application. Thus users can do their archiving from the context of the end-user

application they are working in, without having to turn to a separate archiving tool, and without requiring manual double entry by archivists. This has obvious usability and cost benefits.

Each archiving source gathers as much metadata about the document as it can, and represents it using its own vocabulary. The document is then posted to the CMIS interface, where the CMIS server translates the metadata to the vocabulary used by the archive.

In addition, the metadata is automatically enriched. For example, if the document is tagged with the URI of an electricity meter, the CMIS server will automatically add the URI of the customer currently owning that meter. The metadata translation and enrichment is configured using an RDF vocabulary annotating the CMIS metadata vocabularies. The enrichment code is thus entirely generic, and has no built-in knowledge of the various metadata vocabularies. It also makes the archive clients truly independent of the model used by the archive.

1.4 Ontology

The core of the ontology is at the moment drawn from the ERP system, and contains typical ERP entities like employee, customer, project, and equipment. The ontology is expressed in RDFS, and uses only a few very basic OWL constructs. No reasoning is done using the ontology.

A simplified view of the ontology is shown in figure 2. The full ontology is considerably larger, and changes as new sources and data are added to the system.

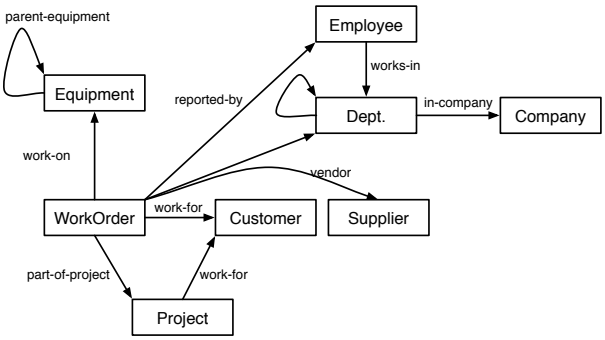

Fig. 2. System ontology

Note that in addition to this core ontology, there are separate ontologies for each source system, subclassed from the core ontology where possible.

2 Principles and Requirements

The architecture of the system has been guided by a few simple principles and requirements described in this section.

2.1 Principles

The interfaces between components should be standards-based, allowing individual components to be changed or replaced without affecting other components. This also enables the use of existing open source or commercial components which support the standards.

The system should be driven by configurations and annotations of the source data, rather than logic defined in code. A corollary is that all mappings should reside in data, not code. Similarly, code should be generic and handle new schema elements correctly without having to be modified. This makes the system much more flexible, and limits the amount of code.

Further, configuration and annotations should be stored in the triple store, rather than in peripheral systems. This makes it easier for developers to make changes without having to be intimately familiar with every component.

Finally, source data should be extracted as-is, and not transformed into a canonical data model for the entire enterprise. Not transforming data dramatically simplifies integrations, and avoids having to "dumb down" the data to the lowest common denominator. Normalization to a common representation can be implemented where necessary as a feedback loop reading source data from the triple store and writing back normalized data.

2.2 Requirements

Archiving, while important and in some cases a legal requirement, is seen by employees essentially as a distraction from their real jobs. It follows that the process must be as simple as possible, and not require users to enter large amounts of metadata.

The system must handle 1000 users, although not necessarily simultaneously.

Initial calculations of data size assumed 1.4 million customers and 1 million electric meters with 30-50 properties each. Including various other data gave a rough estimate on the order of 100 million statements.

The archive must be able to receive up to 2 documents per second over an interval of many hours, in order to handle about 100,000 documents a day during peak periods. The documents would mostly be paper forms recording electric meter readings.

To inherit metadata tags automatically requires running queries to achieve transitive closure. Assuming on average 10 queries for each document, the system must be able to handle 20 queries per second on 100 million statements.

2.3 Technology Choices

To write generic code we must use a schemaless data representation, which must also be standards-based. The only candidates were Topic Maps [ISO13250-2] and RDF. The available Topic Maps implementations would not be able to handle the query throughput at the data sizes required. Testing of the Virtuoso triple store indicated that it could handle the workload just fine. RDF thus appeared to be the only suitable technology.

The canonical approach to RDF data integration is currently query federation of SPARQL queries against a set of heterogeneous data sources, often using R2RML. Given the size of the data set, the generic nature of the transitive closure queries, and the number of data sources to be supported, we considered achieving 20 queries per second with query federation unrealistic.

We therefore had to transfer data from the data sources into the triple store and keep it in sync with changes. Of the open specifications for this SDShare was considered the most suitable.

We chose to use a search engine as the front-end as we considered it better at handling full-text searches of documents, many concurrent user searches, and filtering of search results by access control rules.

2.4 Data Integration

The heart of the data integration is the triple store, in our case Virtuoso. All data in the system, except actual documents and their metadata, is stored in the triple store. In order to reduce the coupling with the triple store product, we only interact with the triple store using SPARQL and SPARQL Update, sent using the SPARQL Protocol. This should theoretically allow us to change triple store without anything more than minor configuration changes in other components.

The data flows between components are implemented using the SDShare protocol.

2.5 The SDShare Protocol

SDShare servers expose data collections, where a collection is a data set defined by the server. It could be an RDF graph internally on the server, but doesn't have to be. The top level of the SDShare interface is the overview feed, which is an Atom feed providing a link to the collection feed for each collection, as shown in figure 3 on the facing page.

The collection feed is the entry point for each collection, and provides two links: one to the snapshot feed for the collection, and one to the fragment feed for the collection. Subscribers to a collection generally record the URL of the collection feed in their configurations.

The snapshot feed contains a list of links to actual snapshots. A snapshot is a representation of the entire collection in some RDF format. Many implementations offer just a single snapshot, which is a service providing a live export of the entire collection to RDF.

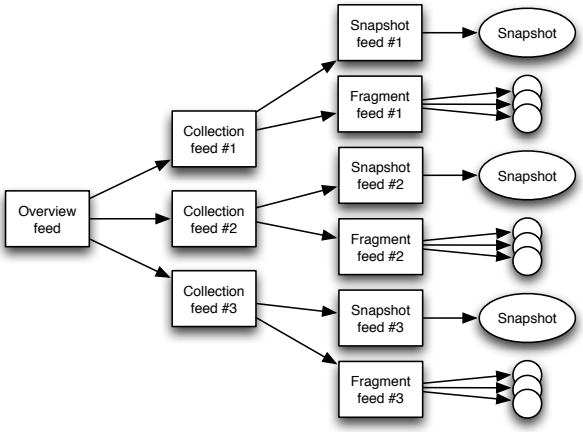

Fig. 3. SDShare server structure

Snapshots serve two purposes: they allow clients to make a local copy before starting to synchronize, and they allow clients to reset their local copy in case there are problems with it.

The fragment feed contains a list of links to fragments. A fragment is a small subset of a collection, typically just all statements where a particular resource is the subject. The fragment feed contains a list of fragments which have changed. By subscribing to it, clients can replicate those changes in their local copies.

The protocol does not inform clients of exactly which triples have changed. Once a resource has changed, the fragment for that resource shows up in the fragment feed, and the client downloads a complete copy of the fragment. The fragment is applied on the client side by simply deleting all statements about the resource in the target graph, and then inserting the new fragment.

The fragment feed often grows very large. In order to avoid having to download the entire feed each time the client polls, a `since` parameter can be added to the request for the fragment feed. The parameter specifies that the client only wishes to see changes after the given time (typically the time of the last change the client has seen).

In addition, servers may page the feed. That is, the fragment feed may be broken into pages, each page providing a `next` link to the next page. Thus clients avoid having to download very large Atom feeds in a single request in cases where there are a large number of changes.

2.6 The SDShare Client

We have developed a generic SDShare client which can be configured with *(Collection feed, SPARQL endpoint)* URI pairs. The client can download a snapshot from the SDShare collection and feed it into the SPARQL endpoint using SPARQL Update. After that, it polls the fragment feed at set intervals for new

fragments. These are also applied to the SPARQL endpoint using SPARQL Update statements.

Adding a new data source thus requires no more than implementing an SDShare server wrapper around the data source, and then adding a new endpoint pair to the configuration. The configuration provides a URI identifying the collection, and this URI is used as the URI of a graph in the triple store. Thus different collections can provide data about the same resources without conflict, as SPARQL Update statements can be used to update the resource in G without modifying it in G'.

The client has support for pluggable backends, and another backend uses a trivially simple HTTP protocol to POST fragments to recipients. This backend is used for recipients which are not triple stores.

2.7 SDShare from the Triple Store

In order to make the RDF data in the triple store available to clients, we expose SDShare feeds from the triple store. This is implemented using an SDShare server framework implemented in Java, which uses SPARQL queries to produce the feeds with `select` queries and the actual snapshots and fragments with `construct` queries. The queries are configurable.

In order to do change data capture we initially added triggers to Virtuoso's RDF data table. These triggers updated a custom table containing the changelog, which was mapped to a virtual graph using Virtuoso's pre-R2RML mapping mechanism, and could thus be queried with SPARQL.

As the system grew, we experienced performance issues with this approach, and so changed the system so that all clients making updates must insert timestamp triples in the triple store when making changes.

2.8 The ERP System

Hafslund uses the IFS ERP system, which is based on an Oracle database. In general, implementing the snapshot part of SDShare on top of Oracle is near-trivial. The difficult part is being able to do change data capture, in order to implement the fragment feed. However, IFS has a history table tracking changes in the database, and the administrator interface can be used to configure which parts of the database have changes tracked. The fragment feed is thus easily implementable through queries against the history table.

This integration has been through a number of iterations, but at the moment it is implemented using the BrightstarDB SDShare server. This is a commercial product which, given a configuration from the relational schema, produces SDShare feeds. It can do change data capture in a number of different ways, including using SQL queries against the change log.

2.9 The CRM Systems

Hafslund at the moment uses two CRM systems: Siebel and Tieto CAB. Here, too, the integration is done at the database level, using the BrightstarDB

SDShare server. The data sets are somewhat larger than from the ERP system, and the databases provide only partial changelogs. This has required using the "last modified" column in cases where data is never deleted. For the remaining cases the BrightstarDB product can compare hashes of database rows against previously stored hashes to see which rows have changed.

The integration with Siebel was completed in just a couple of days. The CAB integration took longer, but in this case the problem was to get the necessary data into the CAB database (as conversion from the system CAB replaces was still in process), and being allowed to access necessary data. That is, the problems were organizational, not technical.

2.10 Sharepoint

The Sharepoint integration reads two SDShare feeds from the triple store, and writes their contents into Sharepoint's taxonomy component (TermStore). The data mapping is a very simple mapping from one RDF property to the TermStore hierarchy, and another to the term labels in TermStore.

The integration keeps copies of the Hafslund organization structure and a hierarchical classification scheme up to date in the TermStore, allowing Sharepoint content to be tagged with these concepts. A separate integration reads the TermStore contents back out as an SDShare feed, so that the internal Sharepoint identifiers for these terms are available in the triple store with sameAs-mappings to the original resources.

2.11 To the Archive

The actual archive system used at Hafslund is Public 360, which takes care of handling basic document metadata, content, versioning, access control, and so on. Public 360 also provides compliance with the Norwegian NOARK archive standard [NOARK5], which is a legal requirement for parts of the Hafslund group.

In order for key metadata required by NOARK to be present in the archive, documents need to be tagged with what Public 360 calls "contacts". These exist in the triple store as employees, customers, and suppliers, and so must be imported into the archive. This was done by configuring a special SDShare feed from the triple store containing only resources of these classes.

For flexibility we wanted to avoid hard-wiring the mappings from the data in the triple store to the Public 360 data model. A mapping vocabulary describing mappings from the RDF data to arbitrary RDF was developed, and is applied by the SPARQL queries used to set up the SDShare feeds. These mappings also filter out data that should not be included.

The Public 360 integration code is thus completely generic, and has no knowledge of the mapping. Instead, RDF statements are mechanically translated into the Public 360 data model, using introspection of the URIs in the RDF to determine which classes and fields in Public 360 to write data to.

2.12 From the Archive

Contacts in the archive are pulled into the triple store, and as their URIs have been stored in the archive, `owl:sameAs` statements to the original resources are included. This allows contact information on archived documents to be translated from the ERP/CRM identifiers for contacts to the archive identifiers for the same contacts.

The SDShare wrapper is implemented against the Public 360 web service API, which provides a log of changes.

3 System Components

3.1 The Search Engine

The search engine used is Recommind. The vendor has added an SDShare connector, allowing Recommind to crawl the SDShare feeds provided by the triple store and the archive to index the entire data set. Which RDF properties to index and display are configured using the Recommind administration GUI.

The user interface application is actually the default search interface of Recommind, heavily customized using JavaScript and CSS. This approach, rather than building a custom application, was chosen in order to save time and cost.

3.2 Archiving

The search engine interface has now been integrated in a number of applications, allowing users to see data from the search engine directly in the application. The integration is done by embedding a generic browser component in the client application.

The integrations also make use of the application context, so that when browsing a particular object in the client application, the web interface displays the entity page for that particular object in Sesam. Thus, when working with a particular customer in a CRM system, the user can switch from the CRM view to the Sesam view to see all relevant information about the customer, including links to duplicates of the customer and information about the same customer in other applications.

At the moment such integrations are provided in IFS, Public 360, Sharepoint, CAB, Siebel, and GeoNIS.

In addition, integrations have been developed that make it possible to send documents directly to the archive from IFS, CAB, and Sharepoint. These work by exploiting functionality for storing documents that's already present in these applications, and picks up the documents for forwarding to the archive. The documents are passed on with their metadata in the source application (including references to related objects in the source application) to the CMIS server.

3.3 The CMIS Server

The CMIS server was implemented using Apache Chemistry OpenCMIS, where we plugged in an implementation of the `createDocument` method. This implementation receives documents, translates metadata to the Public 360 metadata vocabulary, and automatically enriches metadata that's already present.

A mapping vocabulary for CMIS metadata was developed, allowing us to configure things like:

- Mappings from one CMIS property to another.
- Static properties to be inherited from existing values (for example, documents in archive X must have property Y set to Z).
- RDF properties to traverse along to collect additional tags.

In addition, URIs in the metadata identifying resources in the triple store are translated into the URIs for the corresponding resources in the target graph. Incoming metadata may well contain a reference to a customer using its URI from the ERP system, which may need to be translated to the URI of the contact in the archive. This is easily done using SPARQL queries that traverse `owl:sameAs` statements to resources defined in the archive graph (which is the target graph in this context).

In order to inherit metadata by traversing all annotated RDF properties from given tags, repeated SPARQL queries are run to produce transitive closure. Thus, a large number of queries must be run for each archived document.

An example may serve to make this clearer. A subset of the source CMIS metadata might look as shown below. Please note that this is CMIS metadata, represented in the CMIS protocol, and not RDF (CMIS allows URIs as the names of properties).

```
http://.../hummingbird/document-number=3483122
http://.../hummingbird/title=Complaint letter of 2012-07-10
http://.../hummingbird/creation-date=2012-07-13
http://.../hummingbird/references=http://.../ifs/work-order/201013
```

After processing through the CMIS server, it might look as follows:

```
http://.../360/external-id=3483122
http://.../360/archive=3
http://.../360/title=Complaint letter of 2012-07-10
http://.../360/document-date=2012-07-13
http://.../360/tags=http://.../ifs/work-order/201013
http://.../360/tags=http://.../360/project/4882
http://.../360/tags=http://.../360/contact/35823
```

Here we have translated the metadata to the fields used by Public 360, added a static value, and traversed outwards from work order 201013 to find related objects. These related objects have URIs in the IFS graph, but have been translated to the corresponding URIs in the 360 graph, via `owl:sameAs` statements.

3.4 Access Control

There are strict access control rules on many of the documents in the archive, as some contain personal information about individuals and others contain confidential commercial information.

Each individual application has its own acccess control implementation, including the archive system. Access control information is extracted from each system together with the other enterprise data, so that the triple store contains the access control group memberships and settings.

When a user logs in to the search engine their access group memberships are loaded from the triple store. Once the groups have been loaded, Recommind automatically performs real-time filtering of search results based on the user's group memberships.

3.5 Deduplication

Data quality analysis of the ERP system quickly showed that it contains many records representing the same real-world entities (duplicates). This is caused by a number of factors, one being the design of the relational schema, which has one table each for Hafslund group companies, employees, customers, and suppliers. Unfortunately, these four categories overlap considerably, which forces data duplication.

There is also much duplication internally within the customer and supplier tables. This seems to be partly caused by limitations on how payment information is attached to these entities, and partly by careless data entry by users.

The consequences for information retrieval are serious, however. Imagine wanting to find a document about customer when the customer is registered 10 times. To find the document the user is forced to repeat the search for each customer copy. It's clear that this is going to be a problem in practice.

To solve this problem we turned to record linkage techniques [Winkler06]. A quick review of existing software found many tools, but none that seemed to meet our requirements for such a tool, which would have to support:

- Receiving data via SDShare.
- Storing the links found in a database.
- Continuously receiving new data and updating the link database.

In the end we implemented our own record linkage engine, known as Duke [Duke], which solved the problem. Both precision and recall of the deduplication done by this engine appears to be satisfactory for user purposes.

Duke maintains a single table of links in an Oracle database, with time stamps in the table, allowing us to easily expose the links in an SDShare feed. Links are expressed as `owl:sameAs` and `haf:possiblySameAs` statements. The SDShare client thus pulls the discovered links back into the triple store, where they are stored in a separate Duke graph, and displayed by the search engine application.

4 Evaluation

The project has been through a pilot phase, and the implementation phase started in late 2010. The system went into production in the autumn of 2011.

4.1 Performance and Scalability

Triple Store. To give an impression of the scale of the system, table 1 contains an overview of the size of the main graphs in the development environment. Ontology and mapping graphs as well as some graphs with reference data are omitted. The total number of statements in the system is around 630 million, and growing daily. (Hummingbird is the old archive, now replaced by Sesam.)

Table 1. Graph sizes

Graph	Statements
IFS data	5,417,260
Public 360 data	3,725,963
GeoNIS data	44,242
Tieto CAB data	138,521,810
Hummingbird data 1	32,619,140
Hummingbird data 2	165,671,179
Hummingbird data 3	192,930,188
Hummingbird data 4	48,623,178
Address data	2,415,315
Siebel data	36,117,786
Duke links	4,858

Virtuoso has held up to these data sizes very well, running in a 2-node cluster in order to provide failover. It's possible to write queries that run slowly, obviously, but generally performance is good. Virtuoso used to freeze for a few minutes when doing checkpoints, but a configuration change fixed this. As end-users only interact with the search engine the consequences were in any case limited.

Search Engine. Initially, the Recommind search engine was too slow. Searches generally took on the order of 5-10 seconds. The cause was that each RDF property in the triple store was a separate facet, and Recommind scaled poorly with the number of facets. By collapsing these properties into a smaller number of semantically equivalent facets, search times were reduced to less than a second.

However, Recommind cannot index and search at the same time, so searches used to hang for 30 seconds after each indexing. This is a serious problem when indexing runs once every five minutes. Tuning has reduced this issue.

SDShare Synchronization. Synchronization via SDShare has performed very well. Generally, importing a snapshot is faster than transferring the same amount of data via fragments. The performance also varies with the source and sink involved. For our purposes, performance has been adequate. The average time to process a fragment varies from 50 to 450 milliseconds, depending on the source/sink combination.

The SDShare client has been optimized somewhat from the original, naive implementation, to a multi-threaded design where different transfer jobs can run in parallel. In addition, the frontends and backends now use persistent HTTP connections, in order to avoid having to open and close three TCP connections per fragment, as was previously the case.

4.2 Architectural Properties

The system has a number of relatively unusual architectural properties, which in our opinion has contributed greatly to the success of the project:

- Generally, the data integrations are nearly stateless, since the integrations only expose Atom feeds. This greatly simplifies the integrations, and also means they can be deployed on any number of nodes. The SDShare client has a minimal amount of state per integration: the timestamp of the last change.
- The application of SDShare fragments is idempotent, so fragments can be processed more than once with no adverse effects.
- If necessary, we can delete the entire contents of the triple store, and reload everything from the source.
- Uniformity. All data integrations follow the exact same approach.
- Simplicity. Most components in the system (except the CMIS server) are simple, and easy to understand.

4.3 Architectural Flexibility

The architectural flexibility of the system has been proved several times over, in our view.

Changing Components. Perhaps the best example is the extraction of data from the ERP system. Originally, this was implemented using the Ontopia Topic Maps engine, which used a DB2TM component to map the relational data to Topic Maps, and then used the built-in SDShare server to expose SDShare feeds. This worked fairly well, but was a bit slow, and required a big, separate component to be set up and maintained.

Eventually, Ontopia was replaced in favour of Virtuoso's built-in virtual graph mechanism, using the pre-R2RML functionality. The Oracle tables of the ERP system were linked in and mapped, and then queried with SPARQL to produce SDShare feeds using the existing SPARQL-to-SDShare server. This had excellent

performance and worked very well, until we needed to UTF-8 encode URIs to handle primary keys in the ERP system containing non-ASCII characters. After that change performance degraded, and we were not able to fix it.

Finally, we switched to the BrightstarDB SDShare server, which is the component currently used.

In each of these cases, the change had no effect on any other component, except that the SDShare endpoint URI in the SDShare client changed.

Handling Duplicates. When the problem with duplicate resources was discovered we quickly came up with the solution of one SDShare feed into the deduplicator, and another SDShare feed going back. To create the first SDShare feed required no more than a few SPARQL queries in the configuration. The second was likewise trivial to set up.

The entire problem was solved simply by adding a new component, and wrapping it with already existing components. It's difficult to imagine any comparably simple solution with traditional technologies.

4.4 Ease of Development

When the project started, only a few of the developers were familiar with RDF, SPARQL, and SDShare. Generally, this has not been a problem, but for some developers writing generic code that does not have hard-wired data binding has been a bit of a challenge.

4.5 Stability

The stability of the tools throughout has generally been excellent, with the exception of the Public 360 archive system.

Synchronization via SDShare has worked well and been mostly stable. If an SDShare sync process stops for some reason, the only real consequence is that data does not get updated. Once the problem is resolved by admins the data flow starts again, catching up with changes that had not been applied.

It is worth contrasting this with the query federation approach, where the failure of either a data source or the mapping to it risks making the entire system fail or causes part of the data to disappear until the problem is resolved.

4.6 Usability

An interview with a project participant representing the users indicated that users were satisfied with the ability to find content, describing it as "good". The benefits from being able to navigate across system boundaries were then not yet realized, as only the ERP integration was in production at the time.

The users complained about "instability", meaning the indexing problem described in 4.1), and the user interface having some issues with handling of context based on cookies.

The user interface also lacks some functionality users want, such as the ability to attach documents to emails directly from search results.

4.7 Other Aspects

The project won the "Archive of the Year 2012" prize from the Norwegian Archive Council. The rationale was "innovative and strategic use of technology" to "improve data gathering and simplify the use of metadata".

The customer has stated that while the project was expensive, the project has paid for itself through cost savings at the document center [Pretorius2012].

5 Conclusion

Overall, we not only consider this project a success, but have reused the general architecture in other projects with excellent results. Three projects for other customers have already used the same technology, and we expect many more to follow.

Our experience is that using RDF greatly simplifies information integration compared to traditional technologies. We also consider SDShare a key enabler, as it greatly contributes to the simplicity and flexibility of the architecture.

References

[CMIS] Content Management Interoperability Services (CMIS) Version 1.0; OASIS Standard (May 01, 2010), http://docs.oasis-open.org/cmis/CMIS/v1.0/os/cmis-spec-v1.0.pdf

[Duke] Duke; open source software, http://code.google.com/p/duke/

[Winkler06] William, W.E.: Overview of Record Linkage and Current Research Directions. Research report series (Statistics #2006-2) (February 08, 2006); U.S. Census Bureau, http://www.census.gov/srd/papers/pdf/rrs2006-02.pdf

[ISO13250-2] ISO 13250-3: Topic Maps – Data Model; International Organization for Standardization; Geneva, http://www.isotopicmaps.org/sam/sam-model/

[NOARK5] Noark 5 – Standard for elektronisk arkiv; Arkivverket; versjon 3.0 (March 01, 2011), http://www.arkivverket.no/arkivverket/Offentlig-forvaltning/Noark/Noark-5/

[Pretorius2012] Pretorius, J.A.: Enkel arkivering, sikker gjenfinning, og deling av virksomhetskritisk informasjon i Hafslund; oral presentation (November 21, 2012), Video: https://new.livestream.com/accounts/233730/events/1689454

[SDShare] Moore, G., Garshol, L.M.: SDShare - A Protocol for the Syndication of Resource Descriptions; version 1.0 draft (July 10, 2012), http://www.sdshare.org/spec/sdshare-current.html