Kieran Brahney
Information Extraction from Semi-Structured Documents
MSci. Computer Science with Industrial Experience
05/06/2015

I certify that the material contained in this dissertation is my own work and does not contain unreferenced or unacknowledged material. I also warrant that the above statement applies to the implementation of the project and all associated documentation. Regarding the electronically submitted version of this submitted work, I consent to this being stored electronically and copied for assessment purposes, including the Department's use of plagiarism detection systems in order to check the integrity of assessed work.
I agree to my dissertation being placed in the public domain, with my name explicitly included as the author of the work.

Date: _____

Signed: _____

ABSTRACT.

Every hazardous chemical material is required to be accompanied by Safety Data Sheets (SDS). Employing a semi-structured format, they offer information and advice with regard to the safe handling and use of the product. In an effort to separate out specific data items, a variety of information extraction techniques were studied to utilise their best features and apply them to the SDS.

An extensible and scalable Java-based system was designed, implemented and tested. The system supports information extraction using a user-provided lexical specification containing a series of regular expressions. The accuracy of the system was evaluated for each XML element on 40 'unseen' safety data sheets. On structured textual elements, for example phrase codes, the system was able to reach accuracies between 85 – 100%. Unfortunately, where the text was more unstructured, formats deviated wildly and the accuracy dropped as low as 35%.

TABLE OF CONTENTS

WORKING DOCUMENTS

A summary of the working documents for this report can be found at:

http://www.lancaster.ac.uk/ug/brahney/FYP2/

Further links are available from this page to each individual working document.

1 INTRODUCTION

"Every day humans and computer systems generate a staggering 2.5 quintillion bytes of data – so much so that in the past two years alone 90% of the world's data has been produced" (IBM, 2015). Of that, approximately 75% of the data is in an unstructured format. That in itself presents a huge challenge for analysts. The information must first be located, extracted, and then re-formed into a more useful, practical structure than offered in its original amorphous state. Indeed, converting the original sources to a regularised and schematised format allows the information to be more easily searched, integrated and use. For example, consider the task of extracting calendar information from e-mails such that it can be exported to an external calendar. There are a variety of ways of describing who, what and when a calendar event may occur hence making automated information extraction a difficult task.

Information Extraction (IE) is the process of locating and extracting specific information in natural language documents. The field is subject to extensive research across a range of communities, including document analysis, natural language processing (NLP) and machine learning (ML) to name but a few. This work seeks to address the problem of information extraction on semi-structured documents within the domain-specific area of material safety data sheets (SDS). SDS's are regulatory documents for the safe handling, and remedial action within expected scenarios of chemical substances. In June 2015, a new complex regulation to be known as Classification, Labelling and Packaging (CLP) will replace the old system of Dangerous Preparation Directives (DPD). In recent months, the convoluted requirements of the regulation led a number of large industry bodies to produce an XML[1] standard (SDSComXML) for the exchange of SDS between producers, suppliers and users of chemical products (Alliance UG, 2015).

This present work seeks to produce a system which complements their goal by supporting the automatic conversion of SDS into the SDSComXML standard. The system aims to meet the following goals:

- To discover state-of-the-art approaches for information extraction and understand how the system resolve their limitations
- To be extensible and scalable through the use of good programming principles, reducing future maintenance
- To support the extraction and conversion of plain text data into XML, that is present within the first two sections of SDS; with an accuracy of greater than or equal to 95%
- To evaluate the extensibility of the system by comparing its performance on extracting citation data from scientific literature and then comparing the results against the state-of-the-art

In this report, the design and implementation of the aforementioned system are described with respect to the objectives. The report goes on to describe how the system was tested, and the difficulties of parsing unstructured information, particularly in such a short-time frame. The accuracy of the system is then evaluated with respect to its performance on unseen SDS. Furthermore, the extensibility objective is assessed with respect to state-of-the-art methods for extracting citation information from scientific literature.

[1] XML, "Extensible Mark-up Language" is a standard approved by the W3C that is a universal format for data exchange on the Web. XML supports electronic exchange of machine-readable data on the web.

2 BACKGROUND

The Reach Centre (The Reach Centre, 2015) is a leading provider of scientific and analytic services, training, and regulatory guidance in the field of chemical management and risk assessment. The company seeks to capitalise on research that is exploring ways of extracting information from semi-structured documents in order to establish a knowledge base derived from the domain specific of material safety data sheets. Looking further ahead, another aim is for the system to be extensible; this aspect will be compared and evaluated against current practices in related research.

2.1 Existing Environment

The Reach Centre offers a range of online services enabling customers to effectively manage and comply with current and future chemical legislation. The majority of their products are based around SDS, two of their largest being:

- "SDS Manager". A cloud storage system with SDS version tracking capabilities
- "chemtrac". A database of essential information on more than 150,000 chemical substances. The service enables customers to assess compliance and business risk.

All of the company's services are built using PHP, MySQL and frontend web technologies on Linux-based servers. Design considerations are bounded by the systems currently in use within the company.

This work contributes to a new product idea, herein referred to as "SDS Author"; one of its aims is to enable customers to edit SDS via a web portal. A number of years ago, a previous developer at the company had briefly attempted a solution. However, their approach lacked formal structure, used aging technologies, and was nothing more than a couple of hardcoded regular expressions[2]. This present work starts from scratch and hopes to resolve many of the issues present in the original attempt.

2.2 Relevant Work

There exists a vast amount of work in the field of information extraction. This section seeks to compare the range of approaches in order to identify those that are most suitable for the implementation of "SDS Author" in such a short time frame.

2.2.1 Statistical Modelling

In 2001 (Lafferty, et al., 2001) introduced conditional random fields (CRFs), a class of statistical modelling, based on conditional Markov chain models (CMMs). The model is used to encode the relationship between text elements and construct annotations. Since 2001 CRFs have received considerable research interest in the area of information extraction on natural language texts.

Some research (Hetzner, 2008; Peng & Mcallum, 2006) has been carried out regarding the extraction of bibliographical information from literature. In 2006, CRFs were compared against previous best Support Vector Machine (SVM) results for citation extraction and found that the word error rate was reduced by 78% (Peng & Mcallum, 2006). Moreover,

[2] "A regular expression (regex or regexp for short) is a special text string for describing a search pattern. You can think of regular expressions as wildcards on steroids."
Available at: http://www.regular-expressions.info/books.html [Accessed 19 05 2015]

hidden Markov models were used by (Hetzner, 2008) achieving an accuracy of > 90% when tested against the Cora dataset – a heterogeneous set of citation data. However, they noted difficulties in using regular expressions to parse symbol alphabets.

2.2.2 Named Entity Recognition

Named entity recognition (NER), is a sub-field of natural language processing which seeks to classify elements in natural language text into one of several pre-defined classes. There exist a number of techniques for building NER systems, (Eltyeb & Salim, 2014) outlined a state-of-the-art review on chemical NER (CNER) systems. CNER systems aim to recognise chemical entities including trivial names, abbreviations, molecular formula and IUPAC entities.

Many CNER systems are built using statistical model techniques, in particular CRFs (Klinger, et al., 2008; Usié, et al., 2014; Grego, et al., 2009). Generally each of these works achieved an accuracy of approximately 80%. The highest accuracy was achieved by (Klinger, et al., 2008) at 85.6% when tested against a MEDLINE corpus. They did however note that there were difficulties in border detection of long entities especially when occurring in parentheses or enumerations. Chemspot developed by (Rocktäschel, et al., 2012) attempts to combine entity recognition and the classification of chemical in natural language texts using a combined CRF model and a dictionary. They achieved reasonable accuracies of 79%. A similar hybrid approach was applied to CHEMDNER (Krallinger M, 2013) who utilised document indexing to search for chemical entities, reaching an accuracy of 87.39%.

2.2.3 Grammars

There has been increasing research in applying grammars to the field of information extraction, particularly on semi-structured documents. Context-free grammars were compared against CRFs by (Viola & Mukund, 2005) when tasked with extracting bibliography information from literature. They found that CRF's only managed an accuracy of 45.67% whilst the context-free grammar delivered a 72.87% accuracy - an improvement of almost 30%! It is noted that context-free grammars benefit from long range scanning and label dependency which are not possible when using CRFs.

2.2.4 PDF-to-XML

Some research has also worked towards the general task of converting documents into XML. In the early 2000's , attempts were made to convert information from the web into structured XML (Liu, et al., 2000; Meng, et al., 2002) in order to enable more sophisticated queries be performed on otherwise unstructured information. Both adopted a schema-guided approach such that a user would define the type descriptors and mapping rules for the data that should be extracted.

By 2006 their work had been applied by (Déjean & Meunier, 2006) to convert the structure of PDF documents into XML, such that headers, paragraphs, lists and so on can be identified. Their approach involved several steps which included text reconstruction, header/footer detection and paragraph segmentation. Much like other previous works, they noted difficulties in handling tables, figures, captions and so on.

Recently, this work was developed further by (Constantin, et al., 2013) when they developed a fully-automated service for PDF-to-XML conversion of scientific literature service. Their aim was to extract: the title, abstract, authors, text, and bibliography information. Whilst their approach was similar to (Déjean & Meunier, 2006) they also applied additional heuristics to spatial organisation detection. During evaluation they

achieved 77.45% on top-level heading identification and 74.05% on individual bibliography items.

2.2.5 Apache UIMA

Apache UIMA (The Apache Foundation, 2015) is a framework designed to analyse large volumes of unstructured information in order to discover knowledge relevant to the end user. It has been designed to scale, and provides a number of components to analyse the information, for example, language identification, NER and so on.

The framework has been applied to a diverse range of use cases. For example, (Stadermann, et al., 2013) used the framework to extract information from scanned legal contracts using OCR. They analysed their approach against 449 documents and achieved a precision rating of 97% on plain text and 89% on tabular data, recall was slightly worse with 93% on plain text and 81% on tabular data. That being said they noted difficulties in handling the inaccuracies produced by OCR. In another use case description, MedEx (built on Apache UIMA) was used to extract medication information from discharge summarises (Xu, et al., 2010). They achieved accuracies greater than 90%.

2.2.6 Regular Expressions

Another approach to extracting information from structured documents involves the use of regular expressions. These are special sequences of characters for describing a search pattern. They were used by (Xiaoyu, et al., 2010) to extract bibliography information from scientific literature. Their approach used data from the ACM digital library together with a series of regular expressions to identify the authors, title, journal name and so on. Each regular expression was evaluated individually and achieved a range of precision and recall values[3]. For example, the title received 100% precision and 53% recall whilst the journal name achieved 93% precision and 64% recall.

2.3 Limitations

A number of works already mentioned, have used statistical methods to generate named entity recognition systems. In many cases their accuracies were reasonable though far from desirable in a commercial setting (80%). One of the most common reasons for reduced accuracy in these systems is due to lack of context i.e. they rely upon the surrounding text to make informed decisions. Moreover, "SDS Author" hopes to extract the key-value mappings from sub-sections. For example, NER may correctly identify e-mail addresses within the document however "SDS Author" requires knowledge of ownership i.e. supplier one's e-mail address. That being said, NER could supplement "SDS Author" after the key-value mapping has been extracted.

CRF's and CFG's are two commonly described techniques for information extraction tasks. Prior work suggested that CFG's outperformed CRF's and had a number of beneficial features. That being said, CFG's require re-training if the input data changes in some way. The results appear to suggest that either of these approaches would have been ideal for "SDS Author". However, due to the short time frame of this work and the steep learning curve to understand and implement either approach, both were duly noted but not used.

[3] "In information retrieval with binary classification, precision is the fraction of retrieved instances that are relevant, while recall is the fraction of relevant instances that are retrieved."
Available at: http://en.wikipedia.org/wiki/Precision_and_recall [Accessed 24 05 2015].

On the face of it, the Apache UIMA framework is a scalable, extensible system, although provides an empty framework i.e. the developer must build the annotators. The system described in this report requires a generic solution such that it can be applied to multiple use cases. This means that users can ultimately build their own text annotators using expert knowledge and provide it to the system at runtime. The UIMA framework does not support this and would have required a separate application to be developed for each use case. Having said that, "SDS Author" does utilise components from Apache UIMA, for instance, the scalable work queue architecture.

Some research has contributed to the general task of converting PDF's into XML. All the approaches noted difficulties with handling presentation formats such as tables, figures, and erroneous characters when handling PDF documents. That being said, some of their contributions could be useful for future work on "SDS Author", for example (Déjean & Meunier, 2006) developed header/footer detection and (Constantin, et al., 2013) developed spatial organisation detection.

Regular expressions were used to extract reference data from scientific literature (Xiaoyu, et al., 2010). However they noted that often, regular expressions were inaccurate under certain character sets. Moreover, small deviations from the expected input would dramatically decrease the accuracy of the system. Due to their simplicity and the pre-defined structure of sub-sections within SDS's, regular expressions are a suitable approach for "SDS Author" to be implemented in the short time frame. Combined with a user-defined schema guided approach, similar to those described by (Liu, et al., 2000; Meng, et al., 2002), will help meet the extensibility goal of the system.

3 DESIGN

The system aims to accurately support the extraction of meaningful information from a PDF document through an extensible, scalable architecture. In this chapter the design of the system is described, starting with an outline of the overall architecture followed by a more detailed description of the system components.

3.1 System Architecture

Figure 1 depicts the architecture of the system which is composed of 2 major components. The first, an API exposes an interface to the system and defines the actions that external users and systems can perform e.g. fetching, deleting and submitting conversion requests. The second, a worker process, is a standalone program which performs the conversion task. The standalone nature of the program means that more worker processes can be created dynamically, thus allowing the system to scale with its user base. Moreover, multiple worker processes introduce a recovery mechanism whereby if a worker becomes unavailable or doesn't acknowledge task completion another can continue the work.

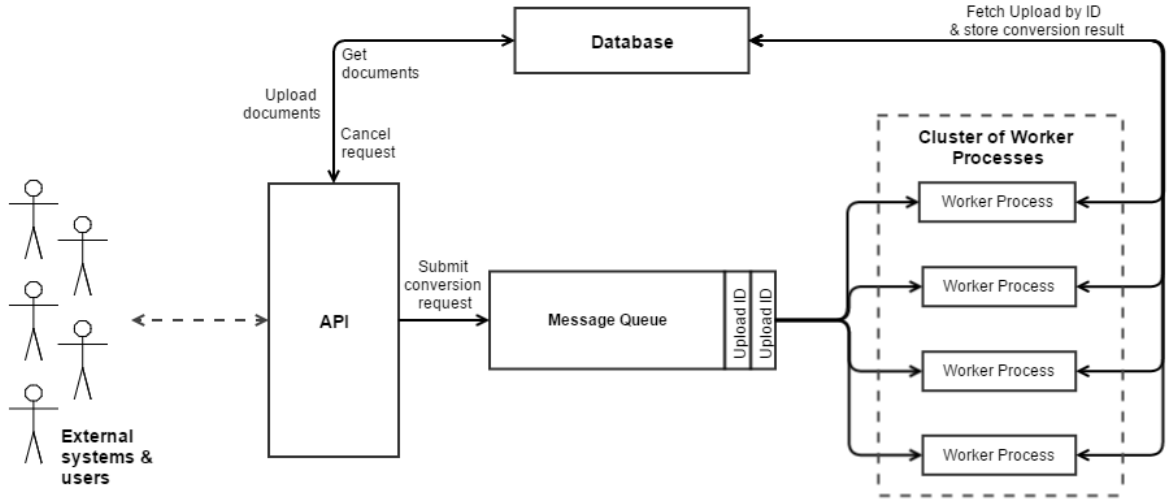


Figure 1 System Architecture

The two major components of the system coordinate their activities via message passing controlled by the message queue. The queue should store a prioritised list of incoming conversion requests so that they can be processed in a first-in-first-out (FIFO) order. When a new conversion request is submitted to the system, the request should be uploaded to the database and added to the back of the message queue. Messages in the queue are then distributed to the first available worker process. Once a worker process has received a message it will begin the document conversion task by fetching the associated message data (files) from the database. Upon completion the message would be acknowledged so that it can be removed from the queue and the worker then becomes available to receive further messages. Furthermore, in order to ensure that conversion requests and their results are available in the future they are stored in a database.

3.2 Database

As required by the company's existing environment, the database should make use of MySQL. Figure 2 (below) depicts the two entities that make up the database and their relationships to one another. The document entity represents the original PDF document; it holds the status of the conversion and if successful, its result. The upload entity represents the normalised state of a document and user-provided specification such that both ultimately represent files. The entity requires that the file hash is included to ensure there are no duplicate binaries in the database hence maximising storage space. Moreover, the filename must be provided so that the binary has its original name when shown back to the user.

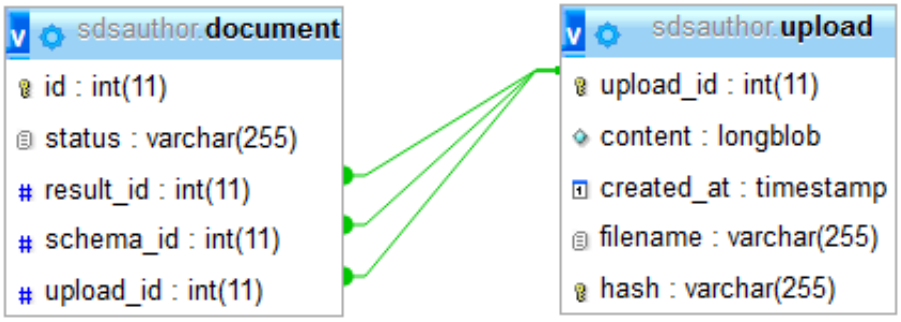


Figure 2 Entity-Relationship Diagram of the two entities making up the system database

There are a number of reasons for storing files in a relational manner. Firstly, it simplifies access as everything is in one place and hence makes backups easier. More importantly, ACID consistency means that files are always in sync and able to participate in transactions. That being said, (MySQL, 2003) states that storing large (1G+) files in the database can present problems as the engine stores 3 copies of it in various buffers. Therefore, in this instance, because files uploaded to the system should not exceed one or two kilobytes, simplicity and ACID properties have been opted for.

3.3 API

The REST API will provide programmatic access to read and write data to the system. It exposes 4 functions allowing users to submit a new conversion request, cancel a conversion request and fetch the information associated with a past or present request. At the moment users do not authenticate to the API and hence can access any information in the system's database; all responses are available in JavaScript Object Notation (JSON)[4]. The four API functions are as follows.

[4] "**JSON** (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate."
Available at: http://json.org/ [Accessed 09 05 2015].

3.3.1 GET documents

Fetch all of the conversion requests that have been submitted to the system. This will return past and present requests including the PDF document, user-provided specification files, and its system status along with associated result / error messages. See GET documents/:id for more specific details on the return values.

Example Request

```
GET
http://api.localhost/documents
```

Example Response


```
[
  {
    "id": 1,
    "upload": {
      "id": 2,
      "hash": "94949321487213872341234",
      "filename": "Acetic Acid _80 - _90% (ALL GRADES) v.3.pdf",
      "content": [ 0 ],
      "created_at": "Apr 29, 2015 4:49:36 PM"
    },
    "schema": { . . . }
    },
    "status": "ERROR"
  },
  {
    "id": 2,
    "upload": { . . . }
    "schema": {  . . . },
    "status": "COMPLETE",
    "result": { . . . }
  }
]
```

3.3.2 GET documents/:id

This function returns a single conversion request, specified by the id parameter. See GET documents for getting all of the conversion requests that have been submitted to the system.

About Status

The status field is used to indicate the progress of the conversion request through the queue. Any of the following from the enumerated list are possible values:

- CANCELLED: the request was removed by the user before processing
- ERROR: the worker failed to complete the request due to a variety reasons (unsupported pdf version, input validation errors, internal implementation errors)
- IN PROGRESS: the request is currently being processed by a worker
- COMPLETE: the request completed successfully

About Result

- The result field holds an upload record, with the two most important fields being the filename and the content. These allow the end-user to access the XML result of the conversion process.
- If there is no result for the conversion request, for example when there has been an error or the conversion request is in progress the field should be omitted from the result – as seen below.

Parameter(s)

id required	The numerical database ID of the desired conversion request. **Example value:** 123

Example Request

```
GET
http://api.localhost/documents/2
```

Example Response


```
[
  {
    "id": 2,
    "upload": {
      "id": 2,
      "hash": "94949321487213872341234",
      "filename": "Acetic Acid _80 - _90% (ALL GRADES) v.3.pdf",
      "content": [ 0 ],
      "created_at": "Apr 29, 2015 4:49:36 PM"
    },
    "schema": { . . . },
    "status": "ERROR"
  }
]
```

3.3.3 POST documents

This function submits a new document conversion request to the system. Firstly, it should validate that each of the required parameters are of the correct file format (pdf and zip), if not a HTTP 400 error will be returned. On successful upload, the function will return HTTP 200 (Accepted) notifying the user that the request has been received and is in the queue waiting to be processed. The user should then poll the GET documents/:id function to monitor the requests progress.

Parameter(s)

pdf_file required	The multi-part file contents of the PDF document to run the conversion request against

zip_file
required
	The multi-part file contents of a zip file containing the following:
	• Java source code (‘*.java*’) for the desired entities to be extracted from the PDF document
	  ○ All classes, methods and fields **must** be annotated using Java Architecture for XML Binding (Oracle, 2015)
	  ○ At **most** one class **must** be annotated with the *@XmlRootElement* annotation
	• At **most** one JFlex specification file with a file extension matching ‘*.jflex*’ or ‘*.flex*’
	  ○ The class annotated by *@XmlRootElement* must be instantiated and used to store extracted information

Example Request

```
POST
http://api.localhost/documents
```

Example Response

```
[
  {
    "id": 5,
    "status": "QUEUED",
    "message": "Task successfully queued."
  }
]
```

3.3.4 DELETE documents/:id

This function should mark a given conversion request as cancelled, notifying the worker processes to ignore the conversion request and remove it from the queue. In cases where the requested id doesn't exist the function will return HTTP 400 error (Bad Request). Moreover, if the conversion task has already been cancelled it will return HTTP 402 error (Not Modified).

Parameter(s)

id required	The numerical database ID of the desired conversion request. **Example value:** 123

Example Request

```
DELETE
http://api.localhost/documents/2
```

Example Response

```
[
  {
    "id": 2,
    "status": "CANCELLED",
    "message": "Document marked as cancelled."
  }
]
```

3.4 Worker Process

3.4.1 Message Queue

Section 3.1 System Architecture described that a central queueing mechanism will be deployed to coordinate and control the message passing of tasks between the API and worker processes. For example, when a task is sent to the queue it will automatically be stored and later distributed to available worker processes. The mechanism would also control acknowledgements of tasks and hence maintain any task deletions from the queue.

At a high level, there are two types of storage engine appropriate for these tasks: 'message queues' and 'key-value stores'. Message queues are ideal for long-running asynchronous processing tasks and for maintaining a prioritised list of messages. Two important properties of message queues are that messages can be configured to be persistent (permanent) and redelivered until acknowledged. However, once messages have been queued they cannot be modified or deleted - messages are only purged once acknowledged by a consumer. Meanwhile, key-value stores are a more modern alternative to traditional relational databases and provide highly scalable, high performance storage engines with read/write capability on messages. Suited to more traditional database usage, they lack an acknowledgement capability and other handy features that come as standard with Message Queues. They also trade data reliability for better performance.

In light of these points, message queues are better suited to the objectives of this system. There are a number of message queue implementations, RabbitMQ (Pivotal

Software, Inc., 2015), Apache Qpid and Apache ActiveMQ to name but a few. RabbitMQ and Apache Qpid were compared by (Batey, 2014) who found that whilst they provided similar features RabbitMQ was the easiest to configure and also had excellent documentation. Based on these findings, the design decision was made to use RabbitMQ for coordinating message passing between the API and worker processes.

3.4.2 Java PDF Libraries

The worker process will need to employ a PDF library in order to accurately extract text, images, tables and other metadata from the document before it can begin the conversion task. There are a number of off-the-shelf and open-source Java-based PDF libraries for example: PDFBox (The Apache Software Foundation, 2015), PDFxStream (Snowtide Informatics Systems, Inc. , 2015), PDFLib (PDFlib GmbH, 2015) and iText (iText Software Corp., 2015). Each of these libraries were evaluated to measure their support for different versions of PDF, character sets (languages) and their extensibility with respect to parsing complex structures (allowing additional system features in the future).

Table 1 Java PDF Libraries rated out of 10

	PDFBox	**PDFxStream**	**PDFLib**	**iText**
PDF Version Support	10	10	10	10
Ability to extract metadata	5	9	3	6
Support for parsing complex structures e.g. tables	4	10	5	6
Open-source	Yes	No	No	No
Documentation	7	7	5	7
General Accuracy	8	9	9	9
Ease of use	10	10	10	6
	44	55	42	44

Table 1 shows the results of the evaluation performed on the four PDF libraries, where each was evaluated out of 10 against several criteria. The results indicate that PDFxStream performed the best. This is primarily an off-the-shelf solution however they also offer the same software but with the limitation that it is single-threaded. Happily, this meant that the free-version came with all the bells and whistles of the paid version and moreover, generally outperformed the other libraries. Furthermore, the library was extensible and also provided a number of conversion formats out of the box, for example: xml, html and text-based.

Meanwhile, as noted by the evaluation scores, the other libraries were somewhat lagging behind. PDFBox came joint second and generally performed equally well with PDFxStream. However, its major downfall was that whilst it was probably the most extensible of all the libraries evaluated, it lacked a lot of features out of the box. iText was another paid-for solution but without the same finesse as PDFxStream. Again, it lacked a lot of features out of the box and more importantly lacked documentation. It was concluded that the library seemed more suited to building PDF's than extracting information from them. Finally, regarding cost, PDFLib was the most expensive of all the off-the-shelf solutions.

The major caveat however was that it was much harder to configure as it was built upon JNI[5] (Java Native Interface). Moreover, the library was much more restrictive after converting the PDF to their own mark-up language "TETML" which would hence then require further parsing.

3.4.3 Lexical Specifications

In order to meet the objective that the system should be extensible, users are required to build their own specification. In the context of The Reach Centre, a user is defined as a system administrator as this is a manual, time-consuming task for an expert. The specification file outlines how to extract information from the source document by defining a series of rules which adhere to a given syntax. The syntax for the specification file is based on the standard Unix lexical analyser Lex (Lesk & Schmidt, 1975). Recently, this was ported to Java in a library known as JFlex (Klein, et al., 2015).

JFlex is a lexical analyser generator for Java that takes a specification defining a set of regular expressions and corresponding actions. It generates the Java source code that reads input, looks for matching regulation expressions and executes the corresponding action, also called a lexer. The produced lexer is based on deterministic finite automata which are fast, without expensive backtracking. Moreover, the library includes a number of features that are important to the system, for example Unicode support, and high performance.

The specification file should be broken down into 3 sections, identified by a single line containing *%%*:

1. User code, defines package and import statements for the generated Java class
2. Options and declarations, consists of a set of pre-defined options used whilst generating the Java class. For example, setting the class name and any variables. Declarations are used to map regular expressions to a macro or abbreviation.
3. Rules and actions, consists of a series of states containing regular expressions and their corresponding actions (Java code). Lexical states are used to group regular expressions and determine the flow of information as the input is read through the scanner.

[5] "JNI is the Java Native Interface. It defines a way for managed code (written in the Java programming language) to interact with native code (written in C/C++)." Available at: http://developer.android.com/training/articles/perf-jni.html [Accessed 09 05 2015].

Table 2 (below) illustrates a very basic example of a lexical specification for reading a safety data sheet. The lexical analyser would begin in the *YYINITIAL* state and upon matching the *SectionOne* macro regex, would create a datasheet instance. Afterwards, it would move into the *SECTION1* state to find regexes appropriate only to that section of the document. One particularly important property to understand of JFlex is that the regular expressions are matched in the order they occur. Moreover, the expression that matches the longest portion of input until a new line character is used first. Hence, ".*" will match everything, if and only if the *SectionOneOne* does not match.

```
/* User code */

%%
/* Options & Declarations*/

%class SDS /* Set the class name to "SDS" */
%standalone  /* Create a public main method */
%xstate SECTION1 /* Exclusive state, only use expressions declared in the state */

%{
    DatasheetFeed datasheet = null;
%}

SectionOne = (1\.?\s+)?(SECTION 1:\s+)?Identification of the substance\/mixture and
of the company\/undertaking

SectionOneOne = (1\.1\.?)?\s+(GHS )?Product identifier

%%
/* Lexical rules */

<YYINITIAL> {
    {SectionOne} {
         datasheet = new DatasheetFeed();
         yybegin(SECTION1); /* Jump to state named SECTION1 */
    }
}

<SECTION1> {
    {SectionOneOne} { System.out.println("Found section 1.1"); }
    .* { /* gobble everything else */ }
}
```

Table 2 Example lexical specification for reading a Safety Data Sheet

3.5 Summary

To conclude, the system is composed of two major components. The API exposes four functions that external users can perform on the system. Primarily these functions fetch and update records stored in a MySQL database. Conversion requests however are published to RabbitMQ, a third-party message queue software, and automatically distributed to worker processes. The following chapter discusses the implementation of the system.

4 IMPLEMENTATION

Earlier, it was noted that the system was split into two main components, an API, and worker processes which coordinate their activities through a message queue provided by RabbitMQ. Both components of the system were built using the Java Spring Framework (Pivotal Software, Inc., 2015) which builds upon the functionality provided by the Java Enterprise Edition (JavaEE); an API and runtime environment for developing large, scalable, and reliable network applications. Spring also ensures that the programmer adheres to good coding practices, for example dependency injection, and loose coupling between modules which should reduce future maintenance time. All of these features should be beneficial to meeting the extensibility objective of the system.

4.1 Database

Both components of the system shared the implementation of entity models which outlined methods for accessing and modifying information in the database. Again, in order to meet the extensibility requirement of the system, a common design pattern and programming technique was employed.

Data Access Objects (DAO) is a design pattern that maps application calls to a persistent layer without exposing the specific implementation details of accessing the database (Oracle, 2002). Section 3.2 describes the two existing tables; DAO suggest that their design is mapped to a Java class (see Table 3). An interface then defines the methods to handle the database operation which manipulate the Java object. Different implementations of the interface can then handle the specific logic for accessing each database engine for example flat file, MySQL and so on. The design pattern is particularly important in ensuring loose coupling between parts of the application that should have no awareness of each other and are accessed frequently. This design pattern helps to meet the extensibility aim by ensuring that in the future developers could easily change the database engine without it significantly affecting the code – only a new interface would need to be written.

Table 3 Data Access Objects example entity and interface

Entity

```
public class Upload {

   private int id;
   private String hash;
   private String filename;
   private byte[] content;
   private Date created_at;

   public Upload(String filename,
         byte[] content)  { . . . }

   public int getId() {
      return id;
   }

   . . .
}
```

DAO Interface

```
public interface GenericDAO<T> {

   List<T> findAll();

   T findById(int id);

   Integer insert(T f);

   void update(T f);

}
```

Object Relational Mapping (ORM) is a programming technique for converting data from database systems to object-oriented objects (redhat, 2015). It is commonly used in conjunction with the DAO design pattern providing the implementation for accessing various database engines through a single method. Whilst they remove the flexibility of hand-coded

SQL statements they provide great transparency between storage engines without having to change application logic and hence reduce maintenance.

4.2 API

As noted earlier, the API was built using the Spring framework. The framework exposes a module known as *Spring-boot* which was particularly important to the implementation as it provides an embedded Apache Tomcat (The Apache Software Foundation , 2015) server. This means that the administrator of the system does not have to worry about complex installation procedures with many dependencies.

The API exposes four functions for external users and systems to make use of the system. Below, UML diagrams describe the workflow when interacting with the three main API functions.

4.2.1 GET documents/:id

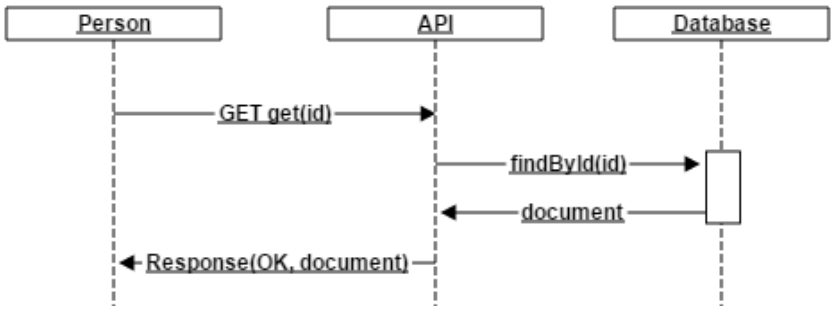


Figure 3 UML Sequence diagram illustrating the flow of information on fetching a request from the database

Figure 3 illustrates one of the most common actions performed on the system: fetching a document via its database id. The action allows a user to fetch up-to-date information about a given document stored in the database. Users may choose to poll this function on a regular basis to monitor the progress of the request through the queue and later fetch the results once marked as complete.

4.2.2 POST documents

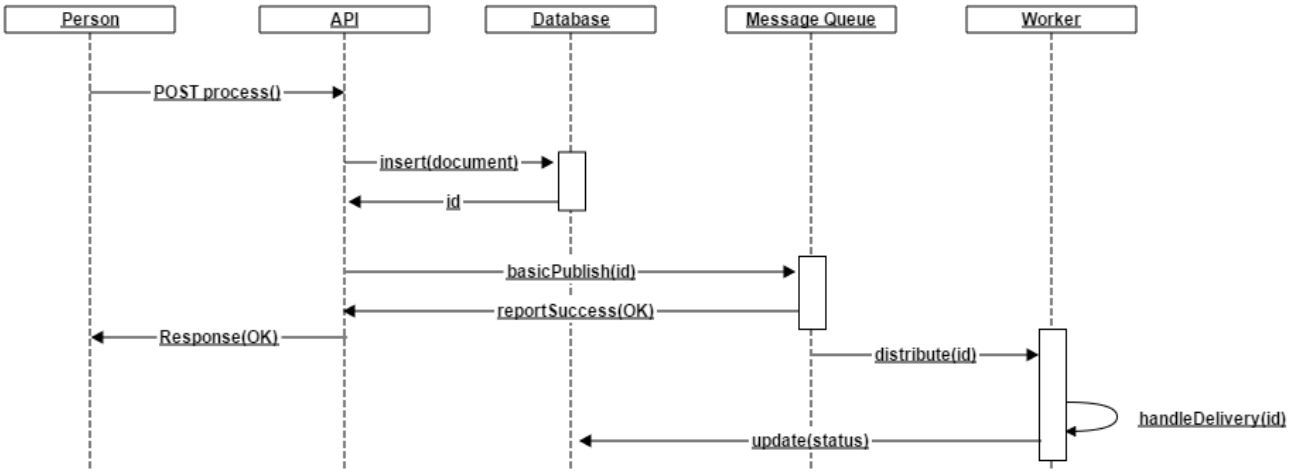


Figure 4 UML Sequence diagram illustrating information flow on a new conversion request

Figure 4 illustrates the primary action performed on the system: submitting a new conversion request. At a high-level this process involves a number of validation checks to ensure that the data meets the requirements specified in the design. If there are no validation errors, the files are uploaded to the database and a message published to the queue containing the inserted database id. The message would then later be consumed by an available worker process. Due to the long-running nature of the processing task, the system immediately returns a success or failure depending on whether it was able to add the task to the queue. The user would then submit GET requests with the request ID to monitor progress.

4.2.3 DELETE documents/:id

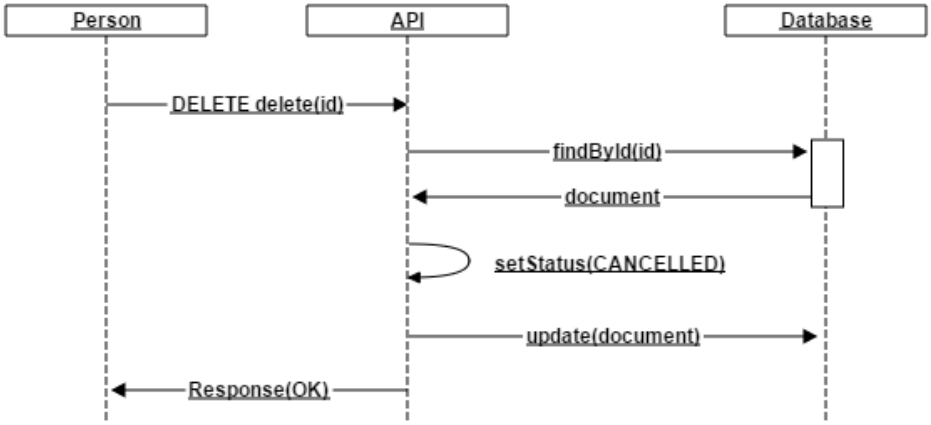


Figure 5 UML sequence diagram illustrating information flow on cancelling a conversion request

Figure 5 illustrates the flow of information required to delete a conversion request. This is very similar to fetching a document from the database. Once the document has been fetched, its status is set to CANCELLED and an update issued to the database. This procedure involves specifying the id of the conversion request that should be removed. In cases where the status of the request has already been marked as "IN PROGRESS" it is not possible to remove the request. This would have required an additional implementation detail that regularly checks the database whilst the worker is processing the request, hence introducing additional overhead on the worker and database.

4.3 Communication Channels

The API and worker processes communicate with each other by connecting to a shared message channel on the RabbitMQ server (described in section 3.4). The channel is configured by the component which first connects to the RabbitMQ server. The configuration involves providing a channel name and whether a hard copy of the queue should be kept. This is important as RabbitMQ is an in-memory implementation, hence once the server is shutdown anything stored in-memory will be lost. Furthermore, the configuration states that RabbitMQ should make use of fair dispatching of messages such that messages are only dispatched to a worker once they have acknowledged completion of the previous message. For compatibility the configuration details are identical across both component implementations.

4.3.1 Publishing messages

When a user would like to submit a new conversion request to the system, the API publishes messages to the RabbitMQ server using its *basicPublish()* method:

```
void  basicPublish(java.lang.String exchange, java.lang.String routingKey,
      AMQP.BasicProperties props, byte[] body)
```
Publish a message

The implementation states that a new message should be sent to the configured queue name containing solely the database id of the inserted 'document' entity (see section 3.2). The `AMQP.BasicProperties` setting also states that the server should redeliver the message another worker in case of failure.

4.3.2 Consuming messages

Messages are dispatched to worker processes automatically by the RabbitMQ server, however they must declare that they would like to consume messages upon connecting. This can be achieved using its *basicConsume()* method:

```
String  basicConsume(java.lang.String queue, boolean autoAck,
        Consumer callback)
```
Start a non-nolocal, non-exclusive consumer, with a server-generated consumerTag.

The implementation of worker processes simply declares that it would like to consume messages from the configured queue name, acknowledge messages by itself and assert that a given class will handle processing of the message. As the auto-acknowledge feature is disabled, the implementation ensures that an acknowledgement is sent upon success or failure. The acknowledgement will delete the message from the queue, thus if there was an error, the user will have to correct the request and resubmit it.

4.4 Worker Process

The worker process component of the system is made up of a number of complex processes. Appendix B: Message Queue Class Diagram illustrates a class diagram showing the flow of object creation in consuming a message and performing the conversion task. Firstly, the Application class, initialises the connection to the MySQL database and RabbitMQ server. The ApplicationReadyListener class is called automatically by the Spring framework once it has initialised. The class initialises the RabbitMQ message channel and database services, hence now being ready to consume messages. Once a message is consumed the MessageConsumer class performs the conversion task.

4.4.1 PDF to Text

Once a message has been received, the document entity is fetched from the database by its ID. The PDF document is then written to a writable location on the file system e.g. /tmp/ on Linux. To extract the text from the PDF document the PDFxStream library, described in section 0 is employed. The implementation makes use of the VisualOutputTarget class which aims to preserve as much of the PDF's text layout as possible – including tabular data.

The PDFxStream library also exposes a number of other OutputHandlers for example XML and HTML. Table 4 illustrates the important differences between the XMLOutputHandler and the desired output of the system. The XMLOutputHandler simply converts blocks of text to XML nodes, whilst the system aims to extract more meaningful information for example the key-value pair: "Emergency telephone" ↔ "01865 407333". One of the main problems with these two output handlers was their detection of text blocks was often inaccurate, sometimes separating individual characters in a header due to different formatting styles.

```
<block height="20.038376" width="361.1928" xpos="100.02228" ypos="213.56015">
    <text>Emergency telephone          :   Emergency only telephone number (open 24
hours):&#13;
number                                  01865 407333 (N.C.E.C. Culham)
    </text>
</block>
```

Table 4 Example output from the XMLOutputHandler

In some rare cases erroneous Unicode characters would exist in the extracted text. To remove some of these irregularities the Java Normalizer class (Oracle, 2015) was used to firstly to convert the characters to standard normalised form. Google Guava (Google, 2015) was then used to identify and remove erroneous characters such as the various types of Unicode white-space and hyphen.

4.4.2 Lexical Analysis

Once the text has been extracted from the PDF, the system executes a number of steps to perform the conversion process as shown in Table 5.

User provided specifications
1. Extract ZIP file to the file system
2. Convert specification file (*.jflex) to Java code
3. Compile *.java files in the extracted directory
4. Execute code produced in (2)
5. Convert instance (4) to XML

Table 5 Pseudo code for performing lexical analysis on the extracted text

Firstly, the zip file provided along with the PDF is written to the file system in a writeable location and its contents extracted. Following the specification outlined in section 3.3.3 the system expects that the zip file contain a user specification file matching JFlex (Klein, et al., 2015) syntax.

Once a file matching an extension type of '.jflex' or '.flex' has been matched in the extracted contents, the library is used to generate a lexical analyser. The source code is written to the same location as the extracted zip contents. Next, '.java' files are searched for

in the extracted directory and compiled using the *javax.tools.JavaCompiler* class. Use of this class requires the Java Development Kit (JDK) to be present on the system.

After successful compilation the compiled lexical analyser is inserted into the current class namespace of the worker process, using the *javassist* (Chiba, 2015) library. This is an important step to ensure that the current classes i.e. MessageConsumer, can access methods defined in the dynamically loaded class. The library is also used to dynamically insert a new method into the generated lexical analyser allowing the system to store an instance of the created class, rather than run the main method which has no return value.

If the class has executed successfully and a reference to its class instance has been saved, one final step remains. In order to convert the result of the lexical analysis to XML, the system requires that the user make use of the Java Architecture for XML Binding toolkit (Oracle, 2015), also known as JAXB. This involves annotating a public variable with the *@XMLRootElement* annotation. The system searches for this annotation in the generated lexical analyser and then simply uses the toolkit to convert the saved class instance into XML.

4.5 Summary

The implementation of the system was built using the Java Spring framework and other third-party libraries to ensure an extensible system was produced. The PDF-to-XML conversion algorithm relies upon three third-party libraries. The first, PDFxStream, converts the PDF document to text. The second, JFlex, converts the user-provided lexical specification into Java source code. And finally, the third, Javassist dynamically executes the produced class providing the conversion result. The following section describes how the implementation was tested.

5 TESTING

This section describes the methods that have been employed to test each individual component of the system. Primarily this includes describing the test cases that have been performed against the technical aspects of the system. Details are also provided with respect to the difficulties of parsing with foreign character sets and how well the system performs. Later, section 6 discusses and evaluates the extraction results in comparison to the state of the art.

5.1 Conversion Algorithm

The API includes three functions which require user input and hence validation. Sections 3.3.2 GET documents/:id and 3.3.4 DELETE documents/:id simply require that the user provide a valid database id. Testing on these functions simply ensured that an error was returned when an invalid database id was given.

Section 3.3.3 POST documents is a much more complicated function requiring additional validation. Table 6 outlines a list of test cases performed against the system along with normal, extreme and erroneous input data. In each case the expected outcome was compared against the actual outcome, when there was a discrepancy the issue was resolved and tested again.

Test case	Input	Expected Outcome
Each request must contain both a PDF and a zip file	PDF file Zip file containing lexical specification and JAXB class	Task successfully queued.
	No PDF file but zip file present	Required MultipartFile parameter 'pdf_file' is not present
	PDF file present but no zip file	Required MultipartFile parameter 'zip_file' is not present
	No PDF or zip file	Required MultipartFile parameter 'pdf_file' is not present
	Pdf_file parameter present but not a valid PDF document (e.g. image) Valid zip file present	Expected ‘pdf_file’ to be of type application/pdf
	Valid pdf file Zip_file parameter present but not valid (e.g. image)	Expected ‘zip_file’ to be of type application/zip
The zip file must contain a syntactically valid lexical	The lexical specification contains valid syntax	Task completes successfully

specification file	The lexical specification uses different options than normal. For example, not specifying %standalone	Task completes successfully
	The file contains a Java syntax error	Task marked "ERROR"
The zip file contains folders	The zip file contains no folders	Task completes successfully
	The zip file contains files nested in multiple folders	Task completes successfully
There should be at most one class annotated with *@XMLRootElement*	1 class annotated	Task completes successfully
	Multiple classes annotated	Task marked "ERROR"
	No classes annotated	Task marked "ERROR"
The system expects the annotated class to be instantiated within the lexical specification as a field	Class instantiated once	Task completes successfully
	Class instantiated multiple times (multiple variables)	Task marked "ERROR"
	No instantiation	Task marked "ERROR"
File upload size	Small file size (<10kb)	Task completes successfully
	Large file size (>5mb)	Task completes successfully

Table 6 A list of test cases for submitting requests to the system along with their expected and actual outcomes

5.2 Variety of inputs

The regular expressions used in the specification to extract information from SDS were built using a corpus of 20 CLP compliant documents. The aim of this was that the corpus would show a common pattern across the majority of safety data sheets and hence the specification should generalise well in the evaluation on unseen documents. Each regular expression was individually tested using a regex debugger tool (Dib, 2015) against the appropriate element from each document in the corpus.

Table 7 (below) shows the variety of the formatting across the corpus of documents. In the majority of safety data sheets we assume a key-value mapping; for example, where the key may be "Company" and the value it's address: "Brenntag UK & Ireland …" Some of the more notable differences in the table include how to extract the key-value pair when the key spans multiple lines of the first column.

Company Identification	Chapel House, Chapel Street, Congleton, Cheshire CW12 4AB
E-Mail (competent person)	registrar@inscx.com ; mw@inscx.com
Emergency Phone No. (GMT)	+44 (1) 782 454 144 Monday - Friday 0900 - 1700
Company :	Brenntag UK & Ireland Albion House, Rawdon Park GB LS19 7XX Leeds Yeadon
E-mail address :	msds@brenntag.co.uk
Emergency telephone number :	Emergency only telephone number (open 24 hours): 01865 407333 (N.C.E.C. Culham)
Lafarge Tarmac Cement Portland House Bickenhill Lane, Birmingham B37 7BQ Technical helpdesk: 0845 812 6323 Emergency telephone number available during office hours (08:30 – 16:00 GMT): Tel +44 (0)845 812 6232 (English language only) Emergency telephone number available outside office hours: None	

Table 7 Individual rows represent the various formatting differences across only one small subsection of the Safety Data Sheet (1.3/1.4)

Appendix C: Emergency phone numbers also shows a subset of possible values from section 1.4 of the SDS. This highlights the difficulty of extracting a telephone number from a value which also contains text. In this instance, the entire value is parsed, non-digits stripped and telephone heuristics used to best-guess a string of digits similar to a phone number. This method ensures that the regular expression isn't tied to a particular country's format of displaying phone numbers.

5.3 PDF versions

One of the most important requirements of the system is that it supports all versions of PDF document. This was verified by looking at documentation when choosing an appropriate PDF library in section 0. Furthermore, as documents can originate from external users a variety of PDF settings were adjusted to see how the system coped. One of the most common settings tested was password-protecting a PDF document. In this case, the PDF library would output an error stating that the document was encrypted and a password required. The status of the request would then be updated to "ERROR" with no further processing taking place.

5.4 Character sets

PDF documents may encode text using any number of predefined character encodings, for example: ASCII, ISO, and UTF to name but a few. Each of these character encoding are important to present the array of characters present in different languages. Until the introduction of Unicode there was no recognised standard, and often no single encoding was adequate for all the letters, punctuation and technical symbols in common use (The Unicode Consortium, 2015).

PDFxStream normalises all extracted text from the PDF document to Unicode, this helps ensure accuracy with respect to special characters in the original document. That being

said, Unicode consists of several different encodings for a single character and sadly the list is continuously growing every year. Table 8 shows a small subset of the 25+ characters that represent whitespace in Unicode; a similar situation exists regarding many other common characters.

Code	Name of the character	Sample	Width of the character
U+0020	SPACE	foo bar	Depends on font, typically 1/4 em, often adjusted
U+00A0	NO-BREAK SPACE	foo bar	As a space, but often not adjusted
U+180E	MONGOLIAN VOWEL SEPARATOR	foobar	No width
U+2000	EN QUAD	foo bar	1 en (= 1/2 em)

Table 8 Subset of the characters that represent whitespace in Unicode (Korpela, 2012)

Testing revealed that character encodings were immediately a problem for the regular expressions. Due to the variety of character codes that ultimately represent the same ASCII (basic) character, the regular expression must make use of special Unicode symbols (Goyvaerts, 2015). For example *\p{Z}* representing any kind of whitespace or *\u00A0* representing the specific character code instead of simply " " (space). In cases where the input character **code** differs from that of the expected value in the regular expression, it will fail to match hence reducing the accuracy of the lexical specification. The problem was further evident for other characters for example dashes (Korpela, 2013). The implementation resolved these problems by replacing the known whitespace and hyphen characters with their ASCII equivalents. However, as there was no complete solution the issue may arise in the future with other characters.

6 EVALUATION

In this section, the system is evaluated using the Levenshtein distance similarity metric to calculate the percentage of documents that are above a given similarity threshold. The metric is applied to the business use case of extracting information from sections 1 and 2 of the SDS's. It is then further applied to the extraction of citation data from scientific literature to demonstrate the extensibility of the system.

6.1 Methodology

The system was evaluated by calculating the percentage of accurately extracted text from the original SDS's. For each of the required XML elements, the system output was compared against its ground-truth counterpart using the Levenshtein string comparison method. Any extracted element found to be above a given similarity threshold was considered a correct match ('a').

		Ground Truth
System outcome	Above threshold	a
	Below threshold	b

Table 9 The number of documents that are above/below the similarity threshold for each XML element

Given Table 9 the percentage accuracy for each XML element is denoted as:

$$Accuracy_e = \frac{(a \times 100)}{(a+b)}$$

6.2 Conversion Accuracy

Using the aforementioned evaluation methodology, the accuracy was reported with respect to SDS's and also citation data to evaluate the extensibility of the system. In each case, the procedure for constructing the ground-truth is described and the accuracy discussed at varying similarity thresholds.

6.2.1 Safety Data Sheets

Due to the lack of an existing dataset, over 1000 SDS's were searched in order to find those which were compliant and hence had the appropriate section and sub-section headings. These was further filtered by selecting SDS's which originated from a variety of companies, as they would typically use the same presentation structure for all SDS's in their product range. In total, 60 SDS's were selected, 20 were used for the construction of the specification file, and the 40 remaining 'unseen' used for evaluation purposes. Each of the SDS's were manually converted into their XML counterpart, representing the expected outcome from the system (the ground truth).

6.2.1.1 Testing Data

Table 10 (below) illustrates the accuracy of the system on the 20 SDS's used to construct the specification file. The total represents the number of SDS's that had a matching element; ground-truth documents with an empty element were omitted. Moreover, the similarity columns, represent the percentage of documents with an XML element that is at least 95% similar to its ground-truth counterpart. It was envisaged that each XML element would achieve an accuracy of 100% at greater than 95% similarity. However, this did not prove to be the case. The challenges affecting the accuracy of some of the results below are described in section 6.2.1.2.

Section 1	Element Name	Levenshtein Similarity			Total
		>95%	>90%	>85%	
1.1	TradeName	100%			20
1.3	PostAddressLine1	85%			20
	PostAddressLine2	85%			20
	PostAddressLine3	85%			20
	Phone	85%	90%		20
	Email	100%			20
1.4	EmergencyPhone	35%		45%	20
Section 2	**Element Name**	**>95%**	**>90%**	**>85%**	**Total**
2.1	ClpHazardClassCategory	85%			7
	ClpHazardPhraseCode	100%			11
	ClpHazardFullText	81%			11
	DpdDsdRiskPhraseCode	100%			12
	DpdDsdRiskFullText	91%	100%		12
2.2	ClpHazardPhraseCode	100%			11
	ClpHazardFullText	72%	90%		11
	ClpPrecauntionaryPhraseCode	100%			13
	ClpPrecauntionaryFullText	23%	30%	53%	13
	DpdDsdRiskPhraseCode	100%			1
	DpdDsdRiskFullText	100%			1
	DpdDsdSafetyPhraseCode	100%			1
	DpdDsdSafetyFullText	0%	100%		1

Table 10 The accuracy of the system when evaluated against the 20 SDS used to build the specification

6.2.1.2 Evaluation Data

Table 11 (below) illustrates how the accuracy of the system changes as the similarity threshold is reduced on 40 ‘unseen’ SDS.

Section 1	Element Name	Levenshtein Similarity						Total
		>95%	>90%	>85%	>75%	>70%	>60%	
1.1	TradeName	69%						39
1.3	PostAddressLine1	45%					50%	40
	PostAddressLine2	57%						40
	PostAddressLine3	40%						40
	Phone	64%						39
	Email	81%						38
1.4	EmergencyPhone	15%		20%				39
Section 2	**Element Name**	**>95%**	**>90%**	**>85%**	**>75%**	**>70%**	**>60%**	**Total**
2.1	ClpHazardClassCategory	71%						7
	ClpHazardPhraseCode	100%						13
	ClpHazardFullText	84%						13
	DpdDsdRiskPhraseCode	95%	100%					23
	DpdDsdRiskFullText	52%	60%		78%	82%	91%	23

2.2	ClpHazardPhraseCode	92%						14
	ClpHazardFullText	42%			57%	64%	85%	14
	ClpPrecauntionaryPhraseCode	85%						14
	ClpPrecauntionaryFullText	14%		28%	42%	57%	78%	14
	DpdDsdRiskPhraseCode	90%						10
	DpdDsdRiskFullText	30%	40%	70%			90%	10
	DpdDsdSafetyPhraseCode	81%					90%	11
	DpdDsdSafetyFullText	45%		63%			90%	11

Table 11 The accuracy of the system when evaluated against 40 'unseen' SDS

Table 10Table 11 illustrate that the accuracy of the system varied quite dramatically between different elements. It is clear that regular expressions resulted in high accuracies when searching for structured text e.g. e-mail addresses and phrase codes. That being said, on the more free-text based elements ("FullText" and "AddressLine") the accuracy decreased markedly. The factors affecting the accuracy of each element are discussed below.

Section 1.1: Trade Name

The evaluation data revealed a 31% decrease in accuracy at 95% similarity, compared to the testing data. The reason for this decline was simply because a number of the evaluation documents exhibited a pattern that had not been seen in the testing documents used to build the specification. This could be resolved by updating the lexical specification.

Section 1.2: Relevant identified uses

Section 1.2 was omitted during the implementation of the specification file due to complications with the SDSComXML standard. In the vast majority of cases, this section contains minimal information, for example "Used in x" or "No information at this time." It is hoped this section could be implemented in the future with assistance from the SDSComXML team.

Section 1.3: Supplier's details

The supplier's details often listed, a company address, telephone number, and e-mail address, or in some cases, multiples of each. Extracting this information was difficult due to the variety of formats used to display the address. For example, some may comma delimit a long string, some may prefix "Street", "Post Code" and so on, whilst others may put each component on a new line. Regular expressions were built to understand address line prefixes and extract the relevant data however, as noted previously, often this was not the case. The accuracy of this section could be improved by integrating named-entity recognition (NER) software to identify phone numbers, e-mail addresses and so on.

Section 1.4: Emergency Phone Number

This section should typically define one or more emergency phone numbers, together with the language spoken and its opening times. This combination made information extraction using regular expressions remarkably difficult. An objective of the system is that it is extensible, and hence able to cope with a variety of telephone number formats. This presents a huge challenge as the formats for telephone numbers vary dramatically from country to country, not to mention writing styles from person to person. The problem was exacerbated when often army times would be used to define the opening hours e.g. 0900 –

1700, see Appendix C: Emergency phone numbers. Again, NER software would have been well suited to complement this task.

Section 2: Classification & Labelling

The system aimed to extract 3 main components from section 2 i.e. the hazard class, phrase codes and the sentence associated with the phrase code. With regards to the accuracy of the hazard class, this was built to compare the extracted word against an enumerated list provided by the SDSComXML standard. That being said, a number of documents often mixed multiple phrases together for example "Org. Perox. CD" to represent "Org. Perox. C" and "Org. Perox. D".

Phrase codes are unique identifiers representing a particular hazard. Their unique identifier made them perfect candidates to be converted to regular expressions and hence resulting in their high accuracy. Unfortunately, the accuracy decreased on a couple of documents which did not explicitly specify the identifier. For example, a header ("H-Statements") followed by "318 - …" rather than "H318 - …". Moreover, similarly to the hazard classes, some phrases contained additional characters, for instance "H318A" and hence were not valid.

Every phrase code typically has an associated full text element describing it, for example "H318" stands for "Causes serious eye irritation". Often these phrase codes can be joined together for example "H318+319+320" and their full text element would describe all three. This made extraction a particularly difficult task as the sentence would often span multiple lines. The system was deliberately not built to cope with these deviations and hence only extracted information up to the new line character. The reason for this was that document headers and footers would often separate sentences spanning multiple lines. These would need to be stripped for extraction to be successful. The results clearly show this, as the accuracy dramatically increased to 100% at around 45% accuracy i.e. everything up to the new line but not anything after it.

Additional Challenges

There were a couple of additional challenges that affected the accuracy of both the testing and evaluation data. Firstly, the PDFxStream library states that there are a number of difficulties with maintaining the texts original appearance (Snowtide, 2015). For example, proportional fonts, justified text and rotated characters can cause issues with determining the correct amount of space between words. Moreover, the library states that one of its main benefits is its support of Unicode, such that it maintains the original characters. This however, presented problems for the regular expressions with invisible white space characters, partial line forwards, and so on, breaking the match. Furthermore, the system does not yet support anything other than single column layouts. Whilst this is a rare presentation for SDS, one of the 60 used in this work utilised a two column layout hence making regular expression extraction difficult.

6.2.2 Bibliography of Literature

To evaluate the extensibility of the system, testing was applied to another use case: extracting citation data from scientific literature. The literature was gathered from Google Scholar and filtered based on two criteria:

1. The document had a single column layout - as noted above, multiple column layouts are not yet supported

2. The referencing format used IEEE or similar. Due to the short time frame of this work, it was important to limit the scale of this second evaluation otherwise it could have easily grown into its own system, as evident by prior work.

In total 10 research papers were randomly selected and manually converted to their XML equivalent. Whilst prior work (Hetzner, 2008; Xiaoyu, et al., 2010) made use of the pre-made datasets, for example the ACM digital library, "SDS Author" requires Java objects to be defined.

Element Name	Average Levenshtein Similarity	Total
Author	48%	10
FullText	54%	10

Table 12 Results of applying the system to extracting authors from bibliography information in scientific literature

Table 12 displays the average Levenshtein similarity across 10 documents for the two extracted elements: authors and the full citation. In both cases, the results are not ideal and worse than prior work. For example, (Xiaoyu, et al., 2010) achieved 74% accuracy when extracting the authors. That being said, they also applied additional heuristics before regular expressions, e.g. reference style identification.

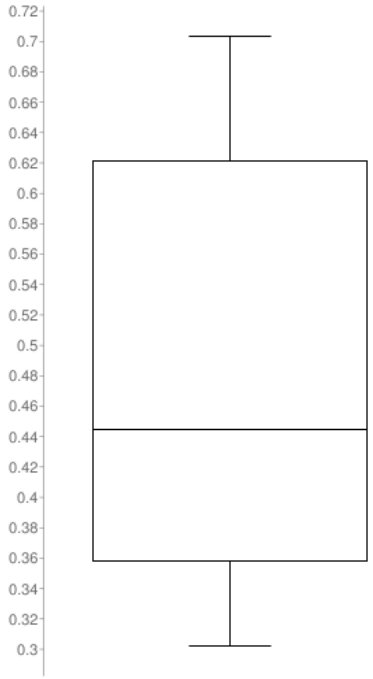


Figure 6 Box plot of the similarity values on the "Author" element

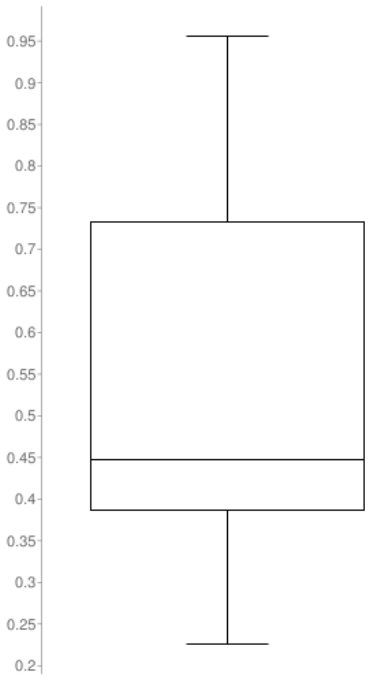


Figure 7 Box plot of the similarity values on the "FullText" element

Figure 6 Figure 7 illustrate the range of similarity values achieved across the 10 documents. It is clear that in some cases the regular expressions worked well however erroneous data easily reduced the similarity. Furthermore, both experienced similar problems to the SDS evaluation, for example text spanning multiple lines and Unicode characters. Given more time, regular expressions could have been built for individual reference styles. Each of which define a fixed structure for the display of citations which could easily map to regular expressions.

6.3 Scalability

To evaluate the scalability of the system, messages were continuously pushed to the API at a rate of 4.2 messages per second, totalling 300 messages with a single worker. Due to the CPU intensive nature of the processing task, the responsiveness of the system varied dramatically depending on the speed of the computers processor and the size of the input documents. On a modern desktop computer the processing speed of each SDS conversion averaged 5.278 seconds, over 40 documents. As more workers are added to the system, the speed with which it is able to process and remove messages from the queue increases.

6.4 Summary

The evaluation has found that the system is able to achieve good accuracy (81 – 100%) on structured text elements that do not split over multiple lines, for example, e-mail addresses and phrase codes that easily map to regular expressions. However, it struggled to extract free-form text elements for example phone numbers and long sentences. The system has proved to be scalable and also generalise to other use cases. That being said, additional heuristics are required to further increase the accuracy, for example header/footer detection and the integration of NER software.

7 CONCLUSION

The brief for this work was to investigate state-of-the-art approaches to information extraction and apply these in the construction of a system to extract information from SDS. The majority of the systems aims were fulfilled however it did have some short-comings which are later discussed in the context of future work.

7.1 Review of the Aims

The system was designed to meet the requirements and specifications gathered from key staff members of the Reach Centre. They were made up of several business goals:

(1) *"To discover state-of-the-art approaches for information extraction and understand how the system can build upon their limitations"*
The work of others involved in this area of research highlighted the complexities that are inherent within it. Their results emphasised the point, whereby none were totally successful in extracting information from unstructured documents. Each approach whether employing CRF's, CMM's, NER's, CRF's, through to using Regular Expressions failed to cope with the irregular format of many of the documents.

(2) *"To be extensible and scalable through the use of good programming principles, reducing future maintenance"*
Good design and programming practices were used including DAO, ORMs and the Spring framework to ensure the system was extensible and robust. Moreover, every step was fully documented to further ease the implementation of any future enhancements by other developers. The system proved to scale with its user base. That being said, the responsiveness of the system was significantly affected by CPU processing power and the ratio of waiting messages to worker processes.

(3) *"To support the extraction and conversion of plain text data present within the first two sections of SDS to XML; with an accuracy of greater than or equal to 95%"*
To a large extent, this was fulfilled, achieving in many instances a near 100% success, whilst in a few instances of unstructured text this fell as low as 15%. This latter case highlights the semi-structured nature of certain sections that then allows 'non-conformity' to creep in. To overcome it, involves a high level of complex amendments to the software, and given the time constraints, this was not possible.

(4) *"To evaluate the extensibility of the system by comparing its performance on extracting citation data from scientific literature and then comparing the results against the state-of-the-art"*
The complexities of information extraction were further apparent when applying the system to another use case. Whilst this did demonstrate the extensibility of the system, the accuracies produced were poor. Prior work applied additional heuristics, for example header/footer identification, that were not possible in such a short time frame.

7.2 Future Work

Given more time, the system would benefit from three additional feature groups:

1. **Pre-processing.** The evaluation noted that multiple column layouts were troublesome when using regular expressions. The system should support the identification of such layouts and convert them to single column.
   PDF documents typically contain a number of additional presentation formats, for example images and tables. Objective decision making ensured that the

PDFxStream library could support the growth of the system. The library supports both the identification and extraction of images and tables. It is envisaged that image extraction could work in a similar manner to MIME e-mails. Whereby a placeholder, representing the image within the text, references its true location on the file system.

2. **Software integration.** It was found that named-entity recognition software could complement regular expressions and hence improve the accuracy of extracting entities from text spread across multiple lines.

3. **Language identification.** The regulation governing SDS's applies across Europe, as a result SDS's may originate in a number of languages other than English. There are multiple approaches to this. For example, translation software could convert the document into English however this may introduce further inaccuracies. Alternatively, different specification files could be produced and language identification software (Lui & Baldwin, 2012) integrated to select the appropriate specification.

7.3 Lessons Learned

Through the work I have been exposed and had to learn a number of new technologies. The Java Spring Framework was without doubt the highest learning curve with the introduction of new programming techniques such as dependency injection and DAO. That being said, the framework provided a rigid structure for producing extensible code and has hence dramatically improved my Java programming skills.

The system made use of a number of additional libraries including Rabbit Message Queue, PDFxStream, Javassist and JFlex. Whilst I had no prior experience of using these technologies I learnt about the importance of objective decision making and taking time to choose appropriate technologies for the business use case. Furthermore, I learnt about the significance of producing well-documented code. Some libraries were much better than others, and those which were not significantly slowed down development. I now realise how poorly documented code affects future developers.

Most importantly the work has taught me the realisation of how projects can easily over-run. Whilst the work was initially ambitious, information extraction is a huge challenge. The work uncovered a number of unforeseen challenges for example parsing Unicode characters. If given the opportunity to do the project again with a longer time-scale I would seek to further compare existing approaches to information extraction for example CRF's and CFG's. The short time scale of this work has been a limiting factor such that there was not adequate time to learn and experiment with new technologies.

7.4 Closing Comments

To conclude, the work aimed to produce an extensible, scalable system that could accurately support the conversion of SDS's into XML. To the best of my knowledge, this is the first attempt at automatic information extraction on SDS's. The system achieved reasonable accuracies on structured elements of text such as e-mail addresses and phrase codes. The accuracy of the unstructured elements could be further improved by refining the regular expressions and integrating other NLP software.

ACKNOWLEDGEMENTS

I sincerely thank Barry Porter for his insightful comments and suggestions. I would also like to thank The Reach Centre for providing me with invaluable work experience and an excellent project concept.

REFERENCES

Alliance UG, 2015. *SDSCom XML - SDS and more.* [Online]
Available at: http://www.esdscom.eu/english/sdscom-xml/
[Accessed 01 03 2015].

Batey, C., 2014. *Rabbit MQ vs Apache QPID - picking your AMQP broker.* [Online]
Available at: http://christopher-batey.blogspot.co.uk/2014/07/rabbit-mq-vs-apache-qpid-picking-your.html
[Accessed 09 05 2015].

Chiba, S., 2015. *Javassist.* [Online]
Available at: http://www.csg.ci.i.u-tokyo.ac.jp/~chiba/javassist/
[Accessed 12 05 2015].

Constantin, A., Pettifer, S. & Voronkov, A., 2013. PDFX: fully-automated PDF-to-XML conversion of scientific literature. In: *Proceedings of the 2013 ACM symposium of Document engineering.* s.l.:ACM, pp. 177-180.

Déjean, H. & Meunier, J.-L., 2006. A system for converting PDF documents into structured XML format. In: *Document Analysis Systems VII.* Berlin: Springer, pp. 129-140.

Dib, F., 2015. *Online regular expressions tester and debugger - JavaScript, Python, PHP and PCRE.* [Online]
Available at: https://regex101.com/
[Accessed 15 05 2015].

Eltyeb, S. & Salim, N., 2014. Chemical named entities recognition: a review on approaches and applications.. *Cheminformatics,* 6(1), p. 17.

Google, 2015. *google/guava.* [Online]
Available at: https://github.com/google/guava
[Accessed 11 05 2015].

Goyvaerts, J., 2015. *Regex Tutorial - Unicode Characters and Properties.* [Online]
Available at: http://www.regular-expressions.info/unicode.html
[Accessed 13 05 2015].

Grego, T., Pęzik, P., Couto, F. & Rebholz-Schuhmann, D., 2009. Identification of chemical entities in patent documents. *Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing and Ambient Assisted Living,* pp. 942-949.

Hetzner, E., 2008. A simple method for citation metadata extraction using hidden markov models.. *Proceedings of the 8th ACM/IEEE-CS joint conference on Digital libraries,* pp. 280-284.

IBM, 2015. *IBM - What is big data?.* [Online]
Available at: http://www-01.ibm.com/software/data/bigdata/what-is-big-data.html
[Accessed 19 05 2015].

iText Software Corp., 2015. *iText Programmable PDF software - iText Software.* [Online]
Available at: http://itextpdf.com/
[Accessed 17 03 2015].

Klein, G., Rowe, S. & Décamps, R., 2015. *JFlex.* [Online]
Available at: http://jflex.de/
[Accessed 11 05 2015].

Klinger, R. et al., 2008. Detection of IUPAC and UIPAC-like chemical names.. *Bioinformatics,* Volume 24, pp. 268-276.

Korpela, J., 2012. *Dashes and hyphens.* [Online]
Available at: https://www.cs.tut.fi/~jkorpela/dashes.html
[Accessed 13 05 2015].

Korpela, J., 2013. *Unicode spaces.* [Online]
Available at: https://www.cs.tut.fi/~jkorpela/dashes.html
[Accessed 13 05 2015].

Krallinger M, L. F. R. O. V. M. O. J. V. A., 2013. Overview of the chemical compound and drug name recognition (CHEMDNER) task. *Biocreative Challenge Evaluation Workshop,* Volume 2.

Lafferty, J., McCallum, A. & Pereira, F., 2001. *Conditional random fields: Probabilistic models for segmenting and labeling sequence data.* San Francisco, 18th International Conf. on Machine Learning.

Lesk, M. E. & Schmidt, E., 1975. *Lex: A lexical analyzer generator,* NJ: Bell Laboratories Murray Hill.

Liu, L., Pu, C. P. & Han, W., 2000. XWRAP: An XML-enabled wrapper construction system for web information sources. In: *Data Engineering, 200. Proceedings. 16th International Conference.* s.l.:IEEE, pp. 611-621.

Lui, M. & Baldwin, T., 2012. *langid. py: An off-the-shelf language identification tool.* s.l., ACL 2012 system demonstrations, pp. 25-30.

Meng, X., Lu, H. & Wang, H., 2002. SG-WRAP: a schema-guided wrapper generator. In: *Data Engineering, 2002. Proceedings. 18th International COnference.* s.l.:IEEE, pp. 331-332.

MySQL, 2003. *TIP: Maximum size of a BLOB.* [Online]
Available at: https://web.archive.org/web/20040410145633/http://www.mysql.com/news-and-events/newsletter/2003-09/a0000000237.html
[Accessed 29 04 2015].

Oracle, 2002. *Core J2EE Patterns - Data Access Object.* [Online]
Available at: http://www.oracle.com/technetwork/java/dataaccessobject-138824.html
[Accessed 17 03 2015].

Oracle, 2015. *Java Architecture for XML Binding.* [Online]
Available at: http://www.oracle.com/technetwork/articles/javase/index-140168.html
[Accessed 12 05 2015].

Oracle, 2015. *Normalizing Text.* [Online]
Available at: http://docs.oracle.com/javase/tutorial/i18n/text/normalizerapi.html
[Accessed 11 05 2015].

PDFlib GmbH, 2015. *PDFlib GmbH.* [Online]
Available at: http://www.pdflib.com/
[Accessed 17 03 2015].

Peng, F. & Mcallum, A., 2006. Information extraction from research papers using conditional random fields.. *Information processing & management42,* Volume 4, pp. 963-979.

Pivotal Software, Inc., 2015. *RabbitMQ - Messaging that just works.* [Online]
Available at: http://www.rabbitmq.com/
[Accessed 17 03 2015].

Pivotal Software, Inc., 2015. *Spring Framework.* [Online]
Available at: http://projects.spring.io/spring-framework/
[Accessed 11 05 2015].

redhat, 2015. *What is Object/Relational Mapping?.* [Online]
Available at: http://hibernate.org/orm/what-is-an-orm/
[Accessed 17 03 2015].

Rocktäschel, T., Weidlich, M. & Leser, U., 2012. Chemspot: a hybrid system for chemical named entity recognition. *Bioinformatics,* 28(12), pp. 1633-1640.

Snowtide Informatics Systems, Inc. , 2015. *PDFxStream - PDF Text, Image and Form Extraction for Java and .NET - Snowtide.* [Online]
Available at: https://www.snowtide.com/
[Accessed 17 03 2015].

Snowtide, 2015. *VisualOutputTarget (PDFxStream API Reference).* [Online]
Available at:
http://downloads.snowtide.com/javadoc/PDFxStream/latest/com/snowtide/pdf/VisualOutputTarget.html
[Accessed 02 06 2015].

Stadermann, J., Symons, S. & Thon, I., 2013. Extracting hierarchical data points and tables from scanned contracts.. *UIMA@ GSCL,* pp. 50-57.

The Apache Foundation, 2015. *Apache UIMA - Apache UIMA.* [Online]
Available at: https://uima.apache.org/
[Accessed 16 05 2015].

The Apache Software Foundation , 2015. *Apache Tomcat - Welcome!.* [Online]
Available at: http://tomcat.apache.org/
[Accessed 17 03 2015].

The Apache Software Foundation, 2015. *Apache PDFBox - A Java PDF Library.* [Online]
Available at: https://pdfbox.apache.org/
[Accessed 17 03 2015].

The Reach Centre, 2015. *The Reach Centre.* [Online]
Available at: https://www.thereachcentre.com/
[Accessed 1 03 2015].

The Unicode Consortium, 2015. *What is Unicode?.* [Online]
Available at: http://www.unicode.org/standard/WhatIsUnicode.html
[Accessed 13 05 2015].

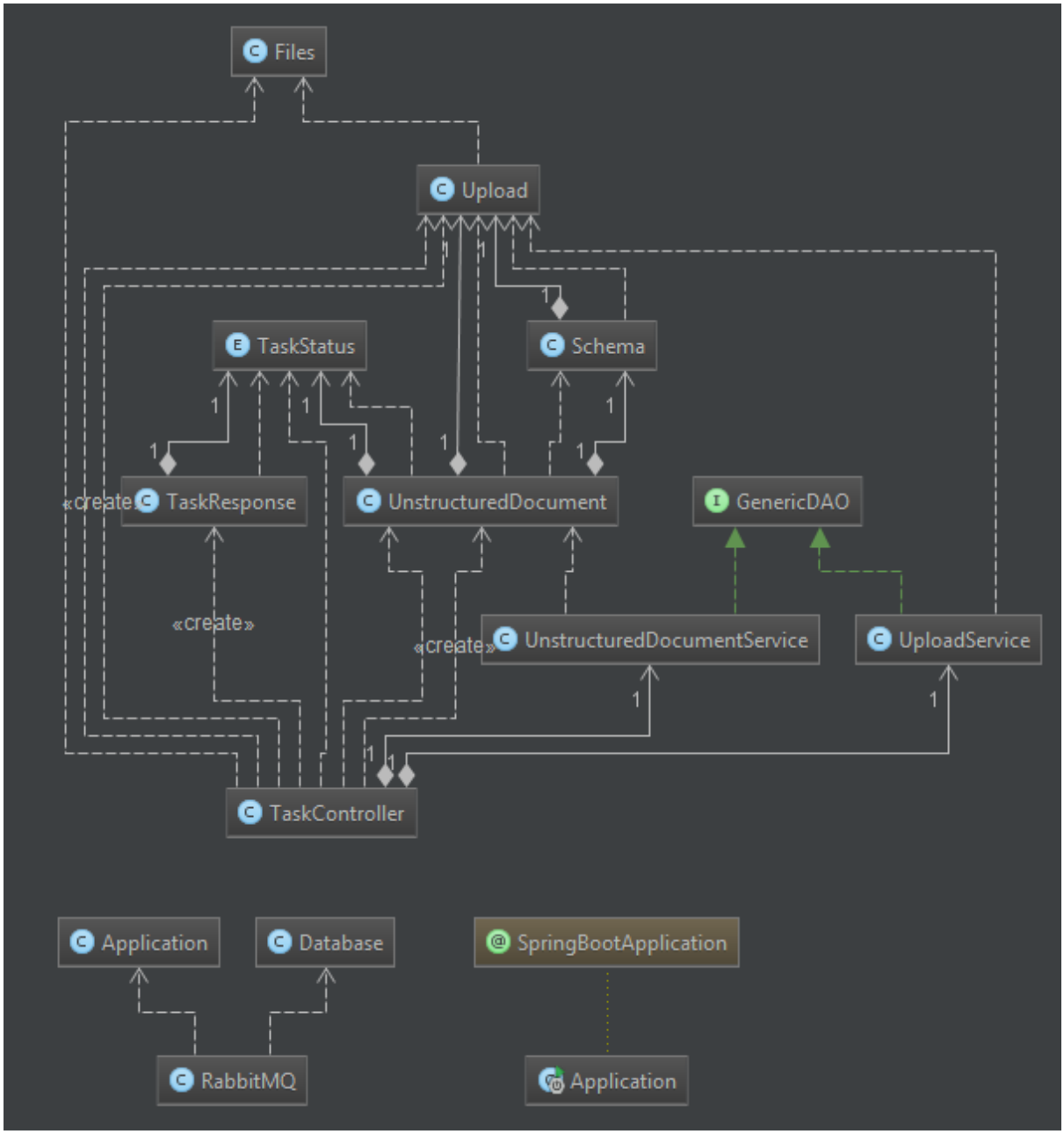
Usié, A. et al., 2014. CheNER: a chemical named entity recognizer. *Bioinformatics*, 30(7), pp. 1039-1040.

Viola, P. & Mukund, N., 2005. Learning to extract information from semi-structured text using a discriminative context free grammar.. *In Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 330-337.

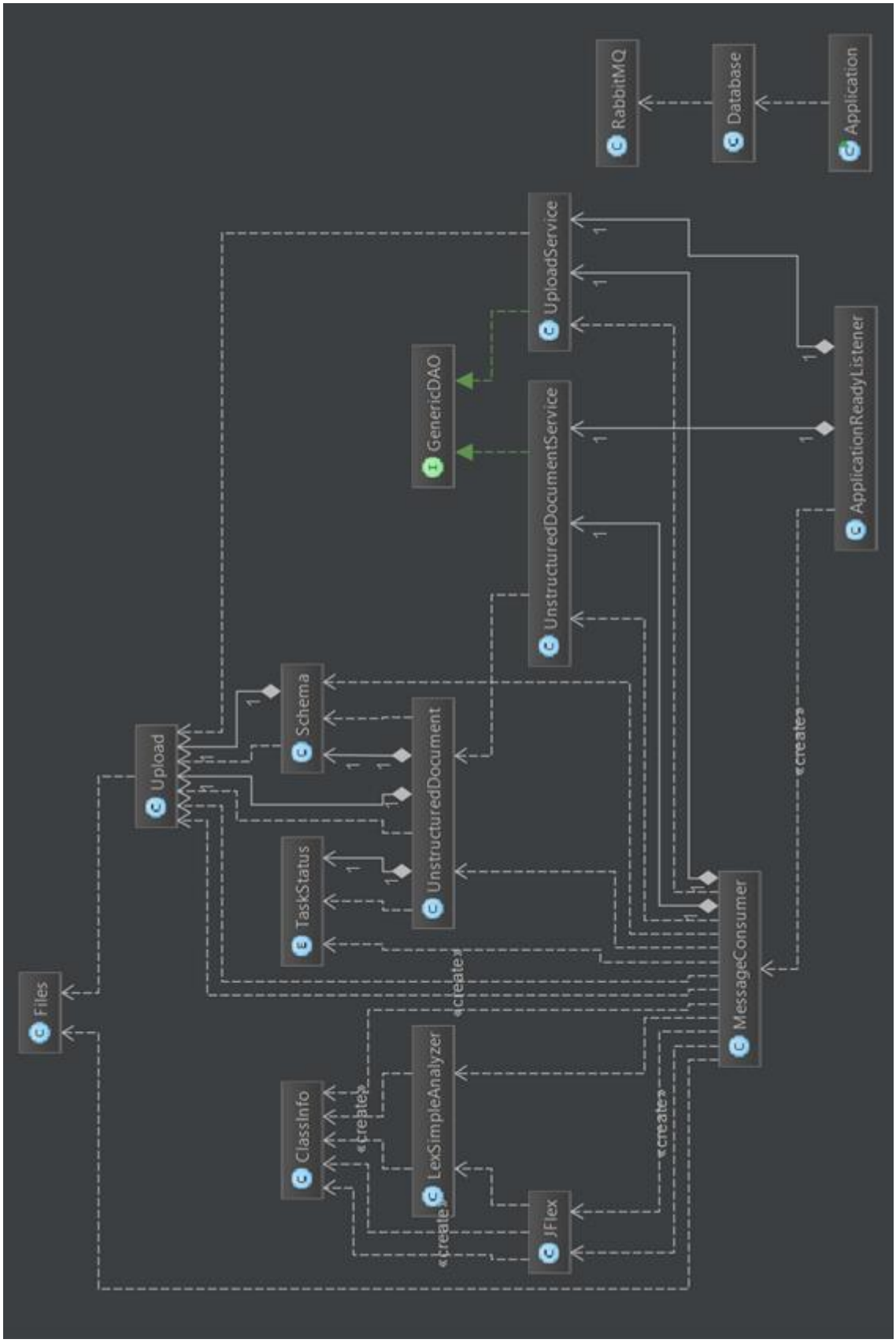
Xiaoyu, T., Zeng, Q., Ciu, T. & Wu, Z., 2010. Regular expression-based reference metadata extraction from the web.. *Web Society (SWS),* Volume 2, pp. 346-350.

Xu, H. et al., 2010. MedEx: a medication information extraction system for clinical narratives.. *Journal of the American Medical Informatics Association* , 17(1), pp. 19-24.

APPENDIX A: WEB SERVICE CLASS DIAGRAM



APPENDIX B: MESSAGE QUEUE CLASS DIAGRAM



APPENDIX C: EMERGENCY PHONE NUMBERS

+44 (0) 1484 514123 09:30 – 1700. GMT, English Spoken.

+44 (1) 782 454 144 Monday Friday 0900 1700 (GMT)

+44 (0) 1453 826661
08:00 – 17:30 Monday - Thursday
09:00 – 16:30 Friday
GMT, English spoken

+44 (0) 1327 356844
Monday - Thursday 8:15 – 17:00 (GMT)
Friday 8:15-12:15 (GMT)

+44 (0) 1934 831 000 (Head Office)
+44 (0) 844 748 0108 (Customer Careline)
Monday - Friday 09:00 – 17:00 (GMT)

+44 (0) 1327 356844
Monday - Thursday 8:15 – 17:00 (GMT)
Friday 8:15-12:15 (GMT)

CHEMTREC
+1 703 527 3887
GMT-5, 08:00-17:00

Manufacturer, Japan
+81 42 554 1203
JST (+9 GMT)

+44 (0) 1925 860999

+44 (0) 1352780861
+44 (0) 7984473602
+44 (0) 7931831452
24 hours, English Spoken.

00 44 (0)151 709 2902
9am-5pm Mon - Fri

0044 (0)1280 822818
09:00 – 17:00 (GMT)

SGS: + 32 (0)3-575-5555
24 hours, multilingual service

Emergency telephone number available during office hours (08:30 – 16:00 GMT):
Tel +44 (0)845 812 6232 (English language only)

IN AN EMERGENCY DIAL 999 (UK only) or 112 (EU)
For specialist advice in an emergency telephone +44 (0) 208 762 8322 (CARECHEM24)

Telephone: +49 180 2273-112

24 hours per day, 7 days per week
Call Infotrac: 1-800-535-5053
Outside N. America: +1 352-323-3500 (collect)

+44-(0) 151-350-4595

+1 800 255 3924

APPENDIX D: PROPOSAL

Information Extraction from Semi-Structured Documents

20/03/2015

Kieran Brahney (32857004)
Barry F. Porter <b.f.porter@lancaster.ac.uk>
Computer Science with Industrial Experience (MSci)

Abstract. Extracting information from semi-structured documents into a regularised format is an important task. It enables the use of sophisticated query tools and hence further integration and re-use of the information. This research seeks to address the problem under the domain-specific area of chemical safety data sheets by converting PDF documents into a structured XML equivalent governed by a schema. The schema will define the blue print for the mapping of content between documents. The work aims to only address the problem on readable PDF documents and support the extraction of plain text, plus tabular data. The expected outcomes of this work are an extensible, scalable system that supports the extraction of content from a safety data sheet into XML using mappings defined in a schema.

1 Introduction

The backbone of the information age is based on the ability to create and share searchable, accessible information. In the last decade there has been rapid growth in the amount of digital information being produced with the introduction of blogs and social media platforms. This has led to an abundance of free-text, content-rich information including structured, unstructured and partially structured domain-specific documents (typically adhering to some regulation). The challenge is that very little of this information has been regularised or schematised to allow it to be searched using more sophisticated query tools which allows the extracted information to be integrated or re-used. For example, one may wish to sort a consortium of documents which are from a certain company, between a time range and by most recent date.

Information Extraction (IE) is the task of locating specific information from natural language documents. It is an area of extensive research interest covering a range of communities, including document analysis, machine learning and natural language processing to name a few. Typically information extraction involves populating a database with values extracted from a document and governed by a schema. A schema gives a blueprint for how the data is organised allowing it to be stored in a regularised format, which is then easily searched, manipulated and re-used. For example, this topic is particularly pertinent in search engines where there are vast amounts of unstructured free-text documents and one would like to find those matching a given query. Google Scholar is an example of this, where bibliography information is extracted from research papers to allow users to easily find (and search on) related articles, cited papers and track revisions.

This research ultimately aims to address the problem of automatically converting semi-structured documents into structured formats governed by a schema. The Reach Centre (1.1 Background) seeks to apply IE techniques within their domain-specific area of material Safety Data Sheets (SDS). SDS are regulatory documents for the safe handling, and remedial action within expected scenarios of chemical substances. They are divided into sections which will be used as a basis for the schema. Extraction of the information from SDS will enable the company to enhance their existing products and also build a whole new range of products by reusing the extracted information. On completion it is hoped that the research will have produced an extensible, scalable system that is able to support the automatic conversion of SDS into XML using a schema. The system should be able to support the companies growing user base and also the high-processing nature of the conversion task.

1.1 Background

The Reach Centre [1] is a leading provider of scientific and analytic services, training and regulatory guidance in the field of chemical management and risk assessment. The company offers a range of online services enabling customers to effectively manage and comply with current and future chemical legislation. Their flagship product chemtrac® is a service containing over 150,000 chemical substances which enables customers to assess product compliance and business risk under a given country's specific regulations. Some of the products more pertinent to this research include:

- "SDS Manager": a service to maintain compliance reports and revisions of SDS
- "SDS Author": a new product idea that would tie into SDS Manager to enable customers to edit their SDS from the web portal (similar to Google Documents) rather than edit the physical document and upload a new copy.

The "EU REACH (Registration, Evaluation, Authorisation and Restriction of Chemicals)" regulation, which came into force in June 2007, is a major driving force within the company. This regulation attempts to improve the protection of human health and the environment from the risks posed by chemicals through the use of SDS. As of June 2015 the regulation is being updated to include a newer standard known as "CLP" (Classification, Labelling and Packaging) – previously "DPD" (Dangerous Preparation Directives). The new regulation states that the document should contain 16 predefined sections each consisting of further predefined subsections, none of which can

be left blank. Beyond this, the presentation of information within each subsection is left to the creators' choice.

1.2 Related Work

Recently, A. Muawia et al. [2] found that there are many different types of approach to IE. The first, natural language processing, is split into two sub-approaches. Named entity recognition (NER), seeks to classify elements in natural language text for example, organisation, location, and person. A statistical modelling technique known as conditional random fields (CRF) was applied by [3, 4] to build a chemical NER model - both found F-measures of 79-80%. Context-free grammars are another approach to NLP, P. Viola et al. [5] found that these performed significantly better than CRFs resulting in an almost 30% increase in accuracy (CFG: 72.87%, CRF: 45.29%). This being said, one problem with CFGs is that they require retraining if there are changes to the input data.

The second, ontology-based IE is defined as the formal specification of a domain-specific conceptualisation. This is therefore defining how to create a schema for documents that may not immediately appear to have an obvious structure. The work performed by [6, 7, 8] describes a range of approaches and techniques for ontology-based extraction. Amongst other types of documents (unstructured, and structured), they found that semi-structured documents typically elicit an ontology from the predefined structure of documents within a universe of discourse.

Research conducted by [9, 10] attempted to convert the structure and content from a PDF document into XML. Both however noted the complexities of dealing with various presentation types (tables, figures, captions). Their work was applied by A. Constantin et al. [10] to scientific literature and achieved 77.45% accuracy on top-level heading identification and 74.05% on individual bibliography items. A similar approach was also applied by [11, 12] who created XML-enabled wrappers for scraping information off the web. X. Meng et al. [12] went one step further to allow the user to define a schema in terms of type descriptors and mapping rules. They concluded that their system can generate wrappers to a "very high accuracy".

A number of IE approaches also make use of the Apache UIMA Ruta framework [13]. J. Stadermann et al. [14] attempted to extract information from scanned legal contracts using OCR technology. They concluded that their system was able to reach a precision rating of 97% on plain text and 89% on tabular data (reduced by OCR inaccuracies), recall was slightly worse with 93% on plain text and 81% on tabular data. H. Xu et al. [15] made use of MedEx[6], which is built-in upon Apache UIMA, to extract structured medication information from discharge summaries; they found F-measures of greater than 90%.

2 Objectives

The research seeks to complete several realistic and feasible objectives:

1. To implement an extensible, scalable system that can accurately convert a readable PDF copy of a Safety Datasheet into XML using a schema by the end of the research period. This objective does not seek to address scanned or unreadable PDF documents.
   This will be measured qualitatively once the research has concluded through the use of software engineering and object oriented programming principles. Moreover, the scalability of the system will be measured by stress-testing the software with various user-base sizes for example: 10, 100, and 1000 per second.

[6] Xu, H., 2013. *MedEx - A tool for finding medication information.* [Online]
Available at: http://knowledgemap.mc.vanderbilt.edu/research/content/medex-tool-finding-medication-information
[Accessed 5 3 2015].

2. To create a schema for the first three sections of the SDS governed by the "CLP" regulation that describes the areas of plain text data to extract. Using this information one is able to determine the majority of the content for the remaining sections. This schema will be used to support the extraction of plain text data, placing it in a regularised format (XML) with an accuracy of greater than or equal to 95.
   This will be measured once the research has concluded using a statistical technique known as the F1-measure (see 3.2 Evaluation Methods).

3. Similar to objective #2 but to support the extraction of tabular data from the first three sections of the SDS to an accuracy of greater than or equal to 90%. Again this objective will be measured once the research has concluded and evaluated against the F1-measure.

4. To make use of detailed logging to support the detection of whether a given SDS complies with the new regulation.
   The nature of the system defines that information should be extracted from all required subsections as defined within the "CLP" regulation. If the system were unable to locate the content of a given subsection then it should generally be considered that it does not comply. This would be measured once the research has concluded and evaluated against a gold standard. To achieve this would involve the creation of a set of documents known to comply and a set of documents that do not comply; the results would then be compared against those from the system.

2.1 Implications

The research intends to contribute a usable, extensible system that accurately supports the conversion of SDS into XML, based on the agreed schema. It is hoped that the extensible nature of the system should enable other industries outside of the field of chemical management to be able to re-use parts for similar use cases. The system would enable SDS to easily be created, stored, manipulated and managed independently of document editors such as Adobe Acrobat or Microsoft Word. This is important because PDF was originally intended as a display format independent of operating system and hardware constraints [16] and not for manipulation / integration. Furthermore, it would mean that there is a standard format for the transmission of these documents electronically.

The work is particularly relevant at the moment with the change in regulation to "CLP". The highly sophisticated requirements of the legislation led some large industry companies to schematise SDS into XML [17]. They share the vision of defining a framework for the structured exchange of material SDS between different IT systems. This research complements their vision by supporting the automatic conversion from PDF to XML. At the moment domain experts manually process the compliance of these documents and track changes by storing physical revisions of the document. A number of the systems at The REACH Centre [1] require metadata, stored within SDS, so extracting this information is vitally important. Furthermore, an automatic conversion system would reduce the need for paper based processing and would also reduce human error during these processes. The schematised format would also offer an easy method to programmatically update SDS and further integration of them into existing software solutions.

3 Methodology

The system will be developed following a waterfall/iterative software development approach. This style of development is particularly well suited to projects of short life, such as this one, and makes it particularly easy to manage with a clear understanding of what and when deliverables will be produced. The methodology is broken into a number of important stages: requirements, design, implementation, testing and maintenance. The requirements section will make use of qualitative interviews and analysing existing documentation (the EU "CLP" regulation"). These methods have

been chosen as only a few people within the company hold the domain-specific knowledge that should be correctly understood in order to implement a solution.

3.1 Data Collection

In order to best design and evaluate the system a large collection of safety Data Sheets are required. Whilst all of these documents should adhere to the regulation it is more than likely that there will be issues with compliance. Analysing these documents should help to identify common terminology which can be filtered into the context-free grammar (see 4 Implementation). The REACH Centre stores over 100,000 clients SDS in-house all of a variety of ages, formats and levels of compliance. A subset of these documents will be requested for testing and evaluation purposes during this work.

3.2 Evaluation Methods

Quantitative analysis will be used to evaluate the accuracy of the system against a gold standard data set. The gold standard data set will be created by a domain-expert by converting a number of SDS into their XML equivalent governed by the "CLP" regulation. The data set should consist of a variety of document styles including those that do and do not comply with the regulation. Amongst this there should also be a variety of quality SDS for example those which have been scanned and those which have not. An F1-measure will be used.

The limited project life and the method of data collection for building the gold standard data set pose some concerns for the above evaluation methods. It currently takes domain-experts within the company approximately one day to accurately check the compliance of an SDS and perform the relevant updates. This is a somewhat similar task as the requirement to build a gold standard data set. One problem with the above evaluation method is that due to the time taken to build the gold standard the evaluation sample may not be large enough and hence not generalise well to a much larger sample. To avoid this problem it may be possible to apply a similar technique to Wittek et al. [18] who used the Apache UIMA Ruta framework to estimate performance on unseen documents while using a small sample size.

4 Implementation

The aim of the research is to design and implement a system that can accurately convert a PDF copy of a Safety Datasheet into its XML equivalent governed by a schema. In order to ensure that the system is extensible and scalable it will be broken down into a number of key components:

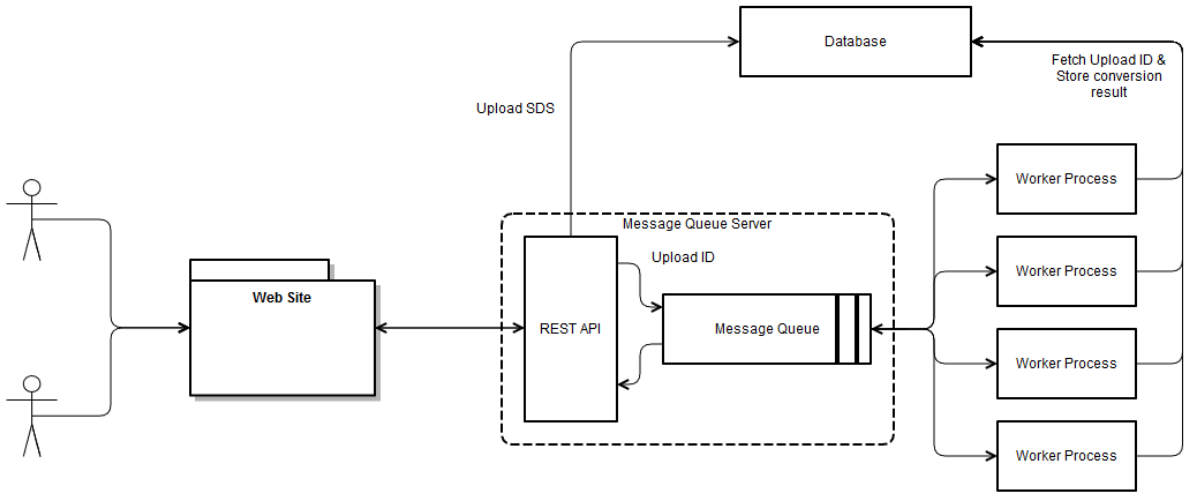


Figure 8 Proposed system architecture

Figure 1 illustrates an example workflow including all the components of the proposed system and the external component or “Web Site” which interacts with the system. It is intended that the system will work independent of other software and hence all interaction will be performed through a REST API. This should ensure that the system is extensible and not tied to any project specific business logic.

The REST API component forms the frontend interface to the system enabling a user to upload an SDS, track the progress of a conversion and also fetch the results. When a user uploads an SDS the physical document will be uploaded and stored in a persistent storage, such as a database, to ensure it is accessible to other company applications. Once the upload is complete the API would forward the conversion request to a message queue which should meet the objective of a scalable system. Worker processes will then fetch conversion requests from the message queue, get the SDS from the database and begin performing all application specific logic. Once a conversion process has been attempted the result would then be stored in the database and available for the client to collect at a later date – for example a regular polling request.

4.1 Tasks

The research consists of several specific tasks as shown in Table 1.

Context	Task	Start Date	End Date	Duration	Timeline
Industrial Placement at The Reach Centre	Requirements Gathering interviews	19/01	21/01	2 days	
	Research & Objective Decision Making on OCR	22/01	30/01	9 days	
	Implementing extensible PDF to text system	02/02	24/02	4 weeks	
	Research & Objective Decision Making on PDF Parser	24/02	24/02	1 days	
	Proposal and report writing (7,000 words)	25/02	13/03	3 weeks	
Dissertation	Producing and gaining agreement to Prototype	20/04	23/04	3 days	
	Creating schema for IE on SDS sections 1 - 3	24/04	29/04	4 days	
	Implementing plain text document parser	30/04	08/05	9 days	
	Implementing tabular document parser	11/05	29/05	3 weeks	
	Testing and Evaluation	01/06	06/05	1 week	

Table 1 Project Gantt chart

During the work placement:

1. ‘Requirements gathering’ ran across 2 days and involved interviews with stakeholders (domain-experts within the company, and the director of IT)
2. ‘OCR Research’ ran across 9 days and involved evaluating various OCR systems against a set of predefined criteria using Objective Decision Making to decide on an appropriate tool
3. ‘Implementation’ ran across 4 weeks and involved creating the base of the system. The system defines a REST API which interacts with a message queue and worker processes to convert a PDF to plain text. The system still requires the plain text to be converted to a structured format using a schema (the research task).
4. ‘PDF Research’ ran 1 day and involved the identification of an appropriate PDF parser for extracting plain text
5. ‘Creation of this proposal and associated report’

The research period will then involve the following tasks:

1. ‘Prototyping’ will run for 3 days and will involve the production of a sample system that meets the specified requirements
2. ‘Creating the schema’ for SDS sections 1-3 will run for 4 days and involve creating a mapping which specifies which subsections to extract from the SDS governed by the “CLP” regulation. For example, you may wish to extract “Product Name” so the schema should know its relative location within the document. This task will require reference to [17].

3. Implementing the ‘plain text parser’ will run for 9 days and marks the start towards achieving the research goals. This task will require a Natural Language Processing library to build a CFG and parse the plain-text document.
4. Implementing the ‘tabular data parser’ will run for 3 weeks as it is the more difficult of the two tasks. This will require similar resources to task 3.
5. Finally ‘testing and evaluation’ will run for 1 week and will involve the construction of a gold standard data set. This task will involve liaising with domain-experts within the company to ensure accuracy when manually converting SDS to their XML equivalent.

4.2 Feasibility

When dealing with SDS there are a few ethical concerns to note. Generally SDS in their final form are publicly available and hence carry no issues. This being said, the research will be handling the SDS of product ingredients and companies certainly wouldn’t want their competitors to know what goes into an end product. Furthermore, one of the objectives of this work involves the detection of potential compliance issues within an SDS. Due to the nature of the chemical industry, companies would want the result of compliance reports to be dealt with in the strictest confidence as any non-compliance would reflect badly on them. In order to mitigate this risk, any documents and logs that are stored within persistent storage should be encrypted.

There are a number of challenges with respect to the feasibility of this research. One problem relates to the dependence on a number of third-party libraries in order to achieve specific components of the system for example: PDF parser, OCR, and the message queue. Relying on third-party libraries causes ones implementation to inherit any security vulnerabilities and also the problem of what to do should a library stop any further production. In order to best manage these risks an objective decision making process will be used to compare and analyse a number of alternatives and the risks associated with each. Moreover, [9, 10] note feasibility difficulties in parsing various presentation formats (tables, captions, figures) in PDFs. This risk has been somewhat mitigated through specific objectives.

5 References

[1] A. Rowntree, “The Reach Centre,” 2015. [Online]. Available: https://www.thereachcentre.com/site/content_home.php. [Accessed 1 March 2105].

[2] A. Muawia, A. Ahmed and M. Himmat, “Information Extraction Methods and Extraction Techniques in the Chemical Document's Contents: Survey,” Journal of Engineering and Applied Sciences, vol. 10, no. 3, 2015.

[3] T. Rocktäschel, M. Weidlich and U. Leser, “ChemSpot: a hybrid system for chemical named entity recognition.,” Bioinformatics, vol. 28, no. 12, pp. 1633-1640, 2012.

[4] A. Usié, R. Alves, F. Solsona, M. Vázquez and A. Valencia, “CheNER: chemical named entity recognizer.,” Bioinformatics, vol. 30, no. 7, pp. 1039-1040, 2014.

[5] P. Viola and M. Narasimhan, “Learning to extract information from semistructured text using a discriminative context free grammar.,” In Proceedings of the ACM SIGIR, 2005.

[6] D. C. Wimalasuriya and D. Dou, “Ontology-based information extraction: An introduction and a survey of current approaches.,” Journal of Information Science, 2010.

[7] P. Buitelaar, P. Cimiano and B. Magnini, “Ontology learning from text: An overview.,” vol. 123, 2005.

[8] A. Maedche and S. Staab, “Ontology learning.,” in Handbook on ontologies, Berlin, Springer, 2004, pp. 173-190.

[9] H. Déjean and J.-L. Meunier, “A system for converting PDF documents into structured XML format,” in Document Analysis Systems VII, Berlin, Springer, 2006, pp. 129-140.

[10] A. Constantin, S. Pettifer and A. Voronkov, “PDFX: fully-automated PDF-to-XML conversion of scientific literature,” in Proceedings of the 2013 ACM symposium of Document engineering, ACM, 2013, pp. 177-180.

[11] L. Liu, C. P. Pu and W. Han, “XWRAP: An XML-enabled wrapper construction system for web information sources,” in Data Engineering, 200. Proceedings. 16th International Conference, IEEE, 2000, pp. 611-621.

[12] X. Meng, H. Lu and H. Wang, “SG-WRAP: a schema-guided wrapper generator,” in Data Engineering, 2002. Proceedings. 18th International COnference, IEEE, 2002, pp. 331-332.

[13] P. Kluegl, M. Toepfer, P.-D. Beck, G. Fette and F. Puppe, “UIMA Ruta: Rapid development of rule-based information extraction applications,” Natural Language Engineering, pp. 1-40, 2014.

[14] J. Stadermann, S. Symons and T. I. , “Extracting hierarchical data points and tables from scanned contracts,” UIMA@GSCL, pp. 50-57, 2013.

[15] H. Xu, S. P. Stenner, S. Doan, K. B. Johnson, L. R. Waitman and J. C. Denny, “MedEx: a medication information extraction system for clinical narratives.,” Journal of the American Medical Informatics Association , vol. 17, no. 1, pp. 19-24, 2010.

[16] Adobe, “PDF Reference sixth edition,” 11 2006. [Online]. Available: http://www.adobe.com/content/dam/Adobe/en/devnet/acrobat/pdfs/pdf_reference_1-7.pdf. [Accessed 09 03 2015].

[17] Alliance UG, “SDSCom XML - SDS and more,” [Online]. Available: http://www.esdscom.eu/english/sdscom-xml/. [Accessed 01 03 2015].

[18] A. Wittek, M. Toepfer, G. Fette, P. Kluegl and F. Puppe, “Constraint-driven Evaluation in UIMA Ruta.,” UIMA@ GSCL, vol. 1038, no. 2013, pp. 58--65.

[19] P. Viola and N. Mukund, “Learning to extract information from semi-structured text using a discriminative context free grammar.,” In Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval, pp. 330-337, 2005.

[20] A. Tharatipyakul, S. Numnark, D. Wichadakul and S. Ingsriswang, “ChemEx: information extraction system for chemical data curation,” BMC bioinformatics, vol. 13, no. 17, p. S9, 2012.

[21] P.-D. Beck, “Identifikation und Klassifikation von Abschnitten in Arztbriefen,” Master's thesis, University of Würzburg, 2013.

[22] H. Xu, “MedEx - A tool for finding medication information,” 2013. [Online]. Available: http://knowledgemap.mc.vanderbilt.edu/research/content/medex-tool-finding-medication-information. [Accessed 5 3 2015].