BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

Design and Implementation of a Modular Benchmarking Framework to Evaluate Information Extraction Quality

Willi Schönborn, 774190
Advisor: Prof. Dr. Stefan Edlich

March 14, 2013

List of Corrections

This thesis is dedicated to my mother who has always given me great support and endured me much longer than I expected.

Acknowledgements

This project would not have been possible without the support of many people.

I wish to thank, first and foremost, Larysa Visengeriyeva for sharing her in-depth knowledge and helping me to bring some sense into the confusion. Many thanks to my advisor, Prof. Dr. Edlich, who read my revisions and offered guidance and support.

A very special thanks goes out to Annelie Sabin, who constantly helped to keep me focused on my thesis. I would also like to thank my friends and coworkers, particularly Sebastian Gröbler and Dennis Bläsing for always showing interest in my work and for proofreading. Appreciation also goes out to Andreas Gohr, for questioning my progress every other day. I would also like to thank my whole family for the support they provided me through my entire life.

Abstract

FiXme Fatal: English abstract

Zusammenfassung

FiXme Fatal: German abstract

Contents

1 Introduction

1.1 Background

Within the age of the Internet and social media sites there is a vast amount of mainly unstructured data being produced on a daily basis. Way too much to handle it in a manual fashion. A lot of research has been done to define, develop and test techniques to extract information from unstructured or semi-structured data sources and transform them into a representation better suited for further analysis. This scientific subfield of Computer Science is called IE.

Since the beginning of IE evaluating the quality of an extractor was always an important factor. It's crucial to have reliable indicators to ensure a continuous improvement in system performances.

1.2 Challenges

IE is missing a set of comprehensive, standard evaluation measures, which makes comparison between different IE algorithms very difficult [60]. Most of the evaluation measures used in current tools are lent from *Information Retrieval*, which usually don't really grasp the inexact nature of IE. Do provide a better way to evaluate IE tools, we need to find out which measurements are available and best suited for IE.

Another challenge faced when evaluating information extractors is that, due to the lack of a proper standards, nearly every available tool defines its own interface and input/output formats. An evaluation tool must therefore define a new format which existing and future systems can be adapted to.

The available evaluation tools for IE tools support usually most of the better known metrics, but they don't support the evaluation of the runtime performance of extractors under testing. The concept of combining the execution and evaluation of information extraction systems into a single platform to measure extraction quality and runtime performance simultaneously is a unique approach. The author did not find any similar approach for information extraction evaluation while conducting research for this thesis.

1.3 Objectives

The goal of this thesis is a formal discussion of known and used performance measures for IE and a working prototype of a highly modular benchmark framework for Java-based platforms to run and test information extraction systems in isolation to measure IE-related performance values, e.g. *precision*, *recall* and *F-Measure*, as well as runtime performance measures, e.g. CPU time and memory consumption. The resulting framework will take external extractors, perform an execution, collect the results, perform the evaluation and return those evaluation results.

The biggest motivator for this thesis is the Database Systems and Information Management (DIMA)[1] group at the TU Berlin, which is currently developing a cloud-based Marketplace for Information and Analytics: *MIA*[2]. The MIA marketplace platform requires such a framework for several of its components, such as extractor ranking and automatic extractor improvements.

Since the framework is planned to be used in a bigger research and development project it has to meet several technical and organizational criteria: The framework has to be developed in an Open Source fashion and released under a business-friendly Open Source license to allow a broad spectrum of researchers and developers to use, modify and improve the framework. To minimize the future development efforts for other researchers and developers, the framework needs to be portable, maintainable and flexible. Portability can be ensured by reyling on the Java programming language and the Java platform in general.

[1] `http://www.dima.tu-berlin.de/`
[2] `http://www.mia-marktplatz.de/`

1.4 Structure

The background knowledge and basics required to put this thesis into context is separated into three chapters: Information Extraction, Formal IE evaluation methodology and Modularity:

Chapter 2 (Information Extraction) contains different definitions of IE, a small discourse about its history, a more or less complete list of the most typical tasks in IE and some information about common IE approaches, current developments and related fields. Formal IE evaluation methodology, chapter 3, shows and discusses current state-of-the-art evaluation techniques and performance measures for information extraction systems and tools. Since the framework is required to be highly modular, we first need to define the term *modularity* and how it affects software design and engineering. Chapter 4 (Modularity) contains different definitions, goals and requirements of modularity as well as a quick overview about modularity in general and Java and the Open Services Gateway initiative (OSGi) service platform in particular.

The chapters 6 Design and 7 Results describe the framework requirements, architecture and implementation steps as well as the result and analysis of the system. The conclusion in chapter 8 will be a critical review of the work done in the course of this thesis as well as an outlook on future work.

2 Information Extraction

Information extraction is a part of the Natural Language Processing (NLP), which focuses its research on the mechanical analysis, processing and generation of natural language. Due to the large amount of information on the internet research in this area is increasingly important to provide access to knowledge and to manage and reproduce the information [64][43].

2.1 Definition

For a better understanding of what IE really is, we should take a look at the following definitions:

> Information Extraction is a technology that is futuristic from the user's point of view in the current information-driven world. Rather than indicating which documents need to be read by a user, it extracts pieces of information that are salient to the user's needs. Links between the extracted information and the original documents are maintained to allow the user to reference context. The kinds of information that systems extract vary in detail and reliability.
>
> **Message Understanding Conference (MUC)**
>
> Chinchor [12]

> Information Extraction refers to the automatic extraction of structured information such as entities, relationships between entities, and attributes describing entities from unstructured sources. This enables much richer forms of queries on the abundant unstructured sources than possible with keyword searches alone. When structured and unstructured data co-exist, information extraction makes it possible to integrate the two types of sources and pose queries spanning them.
>
> **Information Extraction**
>
> Sarawagi [55]

> Information extraction (IE) is the task of automatically extracting structured information from unstructured or semi-structured machine-readable documents. In most of the cases this activity concerns processing human language texts by means of natural language processing (NLP). Recent activities in multimedia document processing like automatic annotation and content extraction out of images/audio/video could be seen as information extraction.
>
> **Information extraction**
>
> Wikipedia [65]

To summarize these definitions one can say that information extraction is concerned with the discovery and therefore the identification of relevant information from large collections of data, and aims to present it in a structured format in order to ensure automatic processing. [42][48][58].

The challenge in information extraction ccording to [30] is the specification of the relevant data. It must be very detailed in order to guarantee an accurate identification. The problem lies in the complexity of natural language. The knowledge can be spread over several blocks and be present in different linguistic representation. The latter occurs, for example, through the use of different names, anaphoric expressions, and similar designations. As part of the extraction, therefore, the existence of the same information regardless of the specific formulation are revealed [1][30][31][43].

2.2 History

The area of text understanding can be considered as the basis for the IE. In this regard, researchers studied methods in the field of Artificial Intelligence, which reproduce the contents of a text in exact form [58][20]. The first application of information extraction occurred in the 1950s, where the information from texts were reduced into a table structure. [32][26][68].

2.2.1 Message Understanding Conferences

The increase in research in the field of IE forced the Defense Advanced Research Projects Agency (DARPA) in the late 1980s to initiate an operation. Thus, the Message Understanding Conferences (MUC) have been launched, aimed at competing implementation and evaluation of IE systems. The DARPA initiated and financed these conferences to encourage research in the field of IE for military and intelligence purposes. Participants of the conferences received test data of a particular domain and a special output format. They then developed IE systems based on these requirements, their performances were compared in terms of unknown documents at the conferences. Manually created templates were used as reference data [33][32].

The conferences were held between 1987 and 1998. The following table lists the domains and the number of training and reference documents of the respective conferences [63][3][15][43]:

	Year	Topic	Number of systems	Traning documents	Reference documents
MUC-1	1987	Marine operations	6	12	2
MUC-2	1989	Marine operations	8	105	25
MUC-3	1991	Terror acts	15	1300	300
MUC-4	1992	Terror acts	17	-	-
MUC-5	1993	Joint ventures, microelectronics	17	-	-
MUC-6	1995	Management changes	17	-	-
MUC-7	1998	Space travel	-	-	-

Table 1: Message Understanding Conferences

The conferences have made a decisive contribution to the development of information extraction, which include the formulation of sub-tasks, metrics and the striving for domain independence and portability of IE systems. The meeting of various research groups and the implementation of systems based on the same task offers enormous opportunities to exchange ideas and to overcome theoretical and paradigmatic differences [13][42].

2.2.2 Automatic Content Extraction

The program of the Automatic Content Extraction (ACE) [1] was launched as a successor to the MUC in 1999. ACE program attempts to focus the research for automatic processing of natural language texts and the development of necessary systems. The field of IE is divided into the following sections [49][41][63][43]:

- **Entity Detection and Recognition (EDR)**
  Identification of entity types and subcategories
- **Relation Detection and Recognition (RDR)**
  Recognition of relationships between entities

[1] `http://www.itl.nist.gov/iad/mig/tests/ace/`

- **Event Detection and Recognition (VDR)**
  Extraction of events and scenarios in which entities are involved

The tasks of the ACE are more complex, as multiple domains are used and also multiple sources of language translation, or the Optical Character Recognition (OCR) need to be analyzed [15].

The results of these conferences will provide the basis of this thesis as described in detail in chapter 3.

2.3 Most typical tasks

The Message Understanding Conferences structured the information extraction into the following sub-tasks due to its complexity [6][41]:

The IE sub-tasks will be explained using the following example document:

> The shiny red rocket was fired near Springfield on Tuesday. It is the brainchild of Dr. Big Head. Dr. Head is a staff scientist at We Build Rockets Inc. [15]

2.3.1 Named Entity Recognition

Named Entity Recognition (NER), also referred to as Name Recognition, Entity Identification or Entity Extraction, is defined as the extraction of known entity names. These include people, organizations, locations, products, date/times and certain numerical expressions [43].

Type	**Value**
`PRODUCT`	rocket
`LOCATION`	Springfield
`DATE`	Tuesday
`PERSON`	Dr. Big Head
`PERSON`	Dr. Head
`ORGANIZATION`	We Build Rockets Inc.

Table 2: Named Entity Recognition example output

2.3.2 Coreference Resolution

Coreference Resolution (CO) is also referred to as Coreference Analysis, Deduplication or Record Linkage. As entities and relationships are extracted from the unstructured source, they need to be integrated into existing databases with repeated mentions of the same information in the unstructured source. The main challenge in this task is deciding if two strings refer to the same entity in spite of the many noisy variants in which it appears in the unstructured source [55].

Example 1: *It* in „It is the brainchild of Dr. Big Head.“ refers to the previously extracted entity *rocket*.

Example 2: *Dr. Head* in „Dr. Head is a staff scientist at We Build Rockets Inc.“ is another spelling for *Dr. Big Head* and therefore also refers to the same entity.

2.3.3 Template Element Construction

Template Element Construction (TE), also referred to as Attribute Extraction, describes the task to associate a given entity with the value of an adjective describing the entity. The value of this adjective typically needs to be derived by combining soft clues spread over many different words around the entity [55].

Attribute	**Target**
shiny red	rocket
brainchild of Dr. Big Head	rocket

Table 3: Template Element Construction example output

2.3.4 Template Relation Construction

Template Relation Construction (TR), also referred to as Relationship extraction, defines to task of extracting relationship information of previously extracted entities. Relationships are defined over two or more entities related in a predefined way. Examples are „is employee of“ relationship between a person and an organization or „is acquired by“ relationship between pairs of companies [55].

The extraction of relationships differs from the extraction of entities in one significant way. Whereas entities refer to a sequence of words in the source and can be expressed as annotations on the source, relationships are not annotations on a subset of words. Instead they express the associations between two separate text snippets representing the entities [55].

Entity	Relation	Entity
Dr. Big Head	works for	We Build Rockets Inc.

Table 4: Template Element Construction example output

2.3.5 Scenario Template Production

Scenario Template Production (ST), also referred to as Event Extraction, tries to extract events that previously extracted entities participate in [15].

Regarding the given example document, ST discovers that there was a rocket launching event on tuesday in which the various entities were involved [15].

2.4 Development and methods of IE

This chapter describes different approaches for the construction of an IE system as well as the current research in the field of information extraction.

There are different approaches for the construction of an IE system which are divided into methods of knowledge engineering and machine learning. It should be noted that an exact categorization is usually not possible because many procedures are a combination of both approaches [57].

The current research in the field of information extraction relates mainly to the extraction of information from text documents and the automatic addition of annotations with the aid of ontologies [43].

2.4.1 Knowledge Engineering

The method of Knowledge Engineering is the manual creation of a grammar by a human expert. Domain knowledge, which is not always available, is necessary to specify extraction rules in which case the method of knowledge engineering can not be applied. Experts must find patterns by inspecting the corpus and produce guidelines according to these patterns [57][63].

The process is usually implemented iteratively. At first one needs to define grammar rules, which are then tested on a training corpus. The rules may be modified depending on the results. The steps are repeated to achieve an acceptable output [3].

2.4.2 Machine Learning

This approach focuses on the extraction based on specific learning processes. It can be made between these methods to the degree of supervision [6][57]:

Supervised learning

This is based on a manually annotated corpus, which contains positive and negative examples of entities. The static feature combinations are used for the extraction of entities and relations. Here, the probability is calculated that it is the extracted data is the desired entities. This range includes learning methods like Hidden Markov Models (HMM) and Support Vector Machines (SVM) [57][59][6].

FiXme Fatal: Probalistic modelling, statistical method

Semi-supervised learning

In this method, a corpus is supplied with a small amount of annotations (seeds). During the application phase the seeds with the best combination of features for customizing existing rules and creating new ones are located and used. This approach is referred to as bootstrapping [6][8].

Unsupervised learning

FiXme Fatal: Purpose for IE, clustering for unsupervised relation extraction

The method of unsupervised learning requires no annotations and manually generated training data. The system will only be given a set of entities whose properties are analyzed. The knowledge gained is the basis for the localization of entities [6][57].

2.4.3 Web information extraction

The rise of the textual sources on the Internet brings an adaptation of existing approaches to extraction. HTML pages are different from text documents, because they contain formatting tags and descriptions. These can, in addition to the page content, contain relevant data. Furthermore, HTML documents contain links to other pages, which may also have relevant knowledge. Because of these challenges, investigations regarding wrappers were launched [20][25][43].

A wrapper is a procedure that identifies data from a source in accordance with special extraction rules. The information within HTML pages are converted into a format explicitly stored for further use. A wrapper must coincide with the dynamic content of the web, manage the change of links and formatting errors. Since a wrapper is limited to one source, research in the area of Wrapper Generation (WG) is initiated [8][20]. The wrapper generation focuses on the integration of multiple sources and counteracts the heterogeneity of the web by using a wrapper library [58][63].

2.4.4 Ontology and knowledge base construction

The idea of the Semantic Web is an extension of traditional content with annotations. The realization requires the creation of annotations, the linking of websites with ontologies and the establishment and management of ontologies. An ontology is a knowledge model that represents concepts and terms, and their relationships. In this context, studies have been started in the field of Ontology-Based Information Extraction (OBIE) which serves the automation of these processes. Unlike traditional information extraction the focus is not alone on the extraction of an entity, but also on the image of an ontology [15][47][64][43].

OBIE faces the following challenges:

- **Identification of instances of the ontology**
  Instances already defined in the ontology need to be found in the documents.

- **Automatic population of the ontology**
  Instances that belong to the concepts of the ontology are added in the correct position.

The advantage over traditional IE is the linking to an ontology which allows a more meaningful storing of the extracted information [47].

2.5 Related fields

IE focuses on the identification and filtering of information relevant to specific a domain. Since IE is a sub-field of NLP, we need to separate it from related fields also concerned with the automatic processing of natural languages.

2.5.1 Information Retrieval

In principle, information storage and retrieval is simple. Suppose there is a store of documents and a person (user of the store) formulates a question (request or query) to which the answer is a set of documents satisfying the information need expressed by his question. He can obtain the set by reading all the documents in the store, retaining the relevant documents and discarding all the others. In a sense, this constitutes 'perfect' retrieval. This solution is obviously impracticable. A user either does not have the time or does not wish to spend the time reading the entire document collection, apart from the fact that it may be physically impossible for him to do so [54].

Information Extraction is not Information Retrieval: Information Extraction differs from traditional techniques in that it does not recover a subset of documents from a collection which are hopefully relevant to a query, based on keyword searching (perhaps augmented by a thesaurus). Instead, the goal is to extract from the documents (which may be in a variety of languages) salient facts about prespecified types of events, entities or relationships. These facts are then usually entered automatically into a database, which may then be used to analyse the data for trends, to give a natural language summary, or simply to serve for on-line access. [27]

2.5.2 Automatic summarization

Automatic summarization involves reducing a text document or a larger corpus of multiple documents into a short set of words or paragraphs that convey the main meaning of the text.

Extractive methods work by selecting a subset of existing words, phrases, or sentences in the original text to form the summary.

In contrast, abstractive methods build an internal semantic representation and then use natural language generation techniques to create a summary that is closer to what a human might generate. Such a summary might contain words not explicitly present in the original.

The state-of-the-art abstractive methods are still quite weak, so most research has focused on extractive methods [28].

2.5.3 Document classification

Document classification is known under a number of synonyms such as document/text categorization/routing and topic identification. Basically document classification can be defined as content-based assignment of one or more predefined categories (topics) to documents. Document classification can be used for document filtering and routing to topic-specific processing mechanisms such as information extraction and machine translation. However, it is equally useful for filtering and routing documents directly to humans [39].

Applications are e.g. filtering of news articles for knowledge workers, routing of customer email in a customer service department or detection and identification of criminal activities for the police or the military [39].

2.5.4 Data Mining

Data Mining, also popularly known as Knowledge Discovery in Databases (KDD), refers to the nontrivial extraction of implicit, previously unknown and potentially useful information from data in databases. While data mining and KDD are frequently treated as synonyms, data mining is actually just one part of the knowledge discovery process, which essentially tries to automatically generate metadata by searching larger volumes of data with the help of certain patterns [69].

2.5.5 Question answering

Question answering is a specialised form of information retrieval. Given a collection of documents, a Question answering system attempts to retrieve correct answers to questions posed in natural language. Open-domain question answering requires question answering systems to be able to answer questions about any conceivable topic. Such systems cannot, therefore, rely on hand crafted domain specific knowledge to find and extract the correct answers [40].

2.6 Summary

This chapter tried to give a small overview of the field of IE by defining the term *information extraction* and giving a small discourse about its history, a more or less complete list of the most typical tasks and some information about common approaches, current developments and related fields. Knowing the underlying ideas and concepts of IE is required to gain a better understanding of how the programs work which will be evaluated later.

3 Formal IE evaluation methodology

FiXme Fatal: More formal approach, see [60]

In the previous chapter the field of information extraction was presented. The process of IE does not only require the extraction itself, but also needs to evaluate the relevance of localized facts for the respective domain. Thus, a quality assessment is required, which focuses on the advancement of development and the juxtaposition of approaches. The evaluation is the formulation of new problems which have to be integrated into the development [43]. The methodology of the evaluation is described in this chapter.

Evaluation has always been an important part in designing, implementing, testing and developing IE systems. It's crucial to have reliable indicators to ensure a continuous improvement in system performances. This chapter aims to provide an overview of the most important and well discussed performance metrics which will be implemented in the framework. For information about the implementation details of the scoring metrics please consult chapter 6 (pp. 35ff.).

FiXme Fatal: Formal definition, description, ...

3.1 Performance measures

A typical way to evaluate an IE system is by using a confusion matrix. This is a well-known technique of counting results. Figure 5 shows a confusion matrix [60].

	Actual class (Observation)	
Predicted class (Expectation)	***tp*** (true positive) Correct result	***fp*** (false positive) Unexpected result
	fn (false negative) Missing result	***tn*** (true negative) Correct absence of result

Table 5: Confusion matrix

For each extracted entity, we have to evaluate if it is correct (and thus a true positive) or not (and thus a false positive). The false negative in the matrix is the number of entities that should have been extracted, but haven't. In IE applications, the true negative is usually not used [60] because we typically do not know what a „true negative“ is. Unlike in document classification, a „bad tuple“ does not exist apriori in a document. It only exists because the extraction system can extract it [35].

Another way to calculate performance measures is based on the notation proposed by Makhoul et al. [46]:

Symbol	Meaning
N	total number of slots in the reference
M	total number of slots in the hypothesis
C	number of correct slots
S	number of substitutions (incorrect slots)
D	number of deletions (missing slots or false rejections)
I	number of insertions (spurious slots or false acceptances)

IE has been, according to [11], traditionally defined as the extraction of information from a text in the form of text strings and processed text strings which are placed into slots labeled to indicate the kind of information that can fill them. To summarize: Slots, in this context, can be understood as the extraction target for entities.

The most used evaluation measures in IE are recall, precision and F-measure. Some additional ones, like Error per response fill and Slot error rate, were proposed by [9] and [46].

3.1.1 Precision

The *precision* (π or P), also called Sensitivity, is defined as the percentage of correctly retrieved data in the hypothesis [6].

$$\pi = \frac{tp}{tp + fp} = \frac{C}{C + S + I}$$

Figure 1: Precision formula

For example, a precision of .9 can be translated to: 90% of the slots in the hypothesis were extracted correctly.

3.1.2 Recall

The *recall* (ρ or R), also referred to as the Positive Predictive Value (PPV), describes the completeness of an extraction, which is determined by the ratio of correctly predicted results to all correct results [6].

$$\rho = \frac{tp}{tp + fn} = \frac{C}{C + S + D}$$

Figure 2: Recall formula

An exemplary recall of .9 can be interpreted as follows: 90% of the slots in the reference were extracted correctly.

Figure 3 shows a graphical interpretation of precision and recall. The relevant items are to the left of the straight line while the retrieved items are within the oval. The red regions represent errors. On the left these are the relevant items not retrieved (false negatives), while on the right they are the retrieved items that are not relevant (false positives). Precision and recall are the quotient of the left green region by respectively the oval (horizontal arrow) and the left region (diagonal arrow) [67].

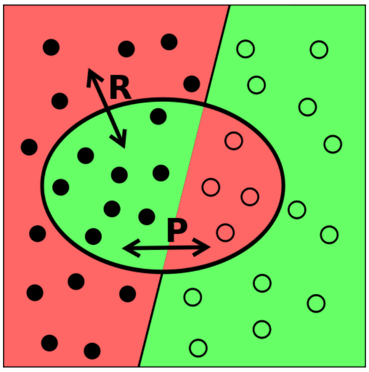

Figure 3: Recall and precision example figure [67]

3.1.3 F-measure

The *F-measure* (F), or balanced F-score or F_1 score, was introduced to combine *precision* and *recall* into a single measure.

$$F_1 = 2 \cdot \frac{\pi \cdot \rho}{\pi + \rho} = \frac{2 \cdot C}{N + M}$$

Figure 4: F-measure formula

Figure 5 shows the more general formula of F_β-score, which contains a parameter β to control the balance between *precision* and *recall*. When $\beta = 1$, F_1 comes to be equivalent to the harmonic mean of π and ρ. If β ¿ 1, F becomes more recall-oriented and if β ¡ 1, it becomes more precisionoriented [56].

$$F_\beta = (1 + \beta^2) \cdot \frac{\pi \cdot \rho}{(\beta^2 \cdot \pi) + \rho}, (0 \leq \beta \leq +\infty)$$

Figure 5: F_β-score formula [10]

F_β measures the effectiveness of retrieval with respect to a user who attaches β times as much importance to recall as precision [54].

Another commonly used formula for the F-measure is shown in 6. In contrast to Rijsbergen's formula, the balance parameter α is balanced when it's set to 0.5.

$$F_\alpha = \frac{\pi \cdot \rho}{(1 - \alpha) \cdot \pi + \alpha \cdot \rho}, (0 \leq \alpha \leq 1)$$

Figure 6: F_α-score formula [46]

3.1.4 Error measure

Since F is a figure of merit, the higher its value the better we consider the performance of the system. We can then define $E = 1 - F_\alpha$ as a corresponding *error measure* [46]:

$$E = 1 - F_\alpha = \frac{S + (1-\alpha) \cdot D + \alpha \cdot I}{(1-\alpha) \cdot N + \alpha \cdot M}, (0 \leq \alpha \leq 1)$$

Figure 7: Error measure formula

3.1.5 Error per Response Fill

The *Error per Response Fill* (*ERR*) is based on the Error measure and removes the deweighting of D and I by simply removing the α weights [9][46].

$$ERR = \frac{S + D + I}{C + S + D + I}$$

Figure 8: Error per response fill formula

3.1.6 Slot Error Rate

The *Slot Error Rate* (SER) was originally proposed by Makhoul et al. [46] and is basically the Error per response fill metric without the I insertion errors in the denominator:

$$SER = \frac{S + D + I}{N} = \frac{S + D + I}{C + S + D}$$

Figure 9: Slot Error Rate formula

3.2 Discussion

Often only the F-measure is reported as the evaluation measure of an IE system. If the same weighting for recall and precision is used in calculating the F-measure, this gives an indication of which system is the better one. However, often it may be important to know the individual recall and precision scores of a system to be able to fully compare different systems. When one system has a recall of 10% and a precision of 90%, this will obtain the same F-measure as a system which obtains a recall of 90% and a precision of 10%, even though both systems are very different. Differences on how a system scores with regards to recall and precision will go unnoticed when reporting only F-measure [60].

For $\alpha = 0.5$, E in 7 reduces to:

$$E = \frac{S + (D+I)/2}{C + S + (D+I)/2} = \frac{S + (D+I)/2}{(N+M)/2}$$

Figure 10: Error measure for $\alpha = 0.5$

The denominator in figure 10 is equal to the average of the number of slots in the reference and in the hypothesis. But the major effect in 10 is the fact that, in the numerator, the deletion and insertion errors are cut (or deweighted) by a factor of two. If the objective is to count all errors, then there is no a priori reason why one should deweight deletions and insertions in this manner. [46] argues that, by simply using F as our performance measure, we are implicitly discounting our overall error rate, making our systems look like they are much better than they really are.

A possible solution to the problem described above is provided by the error measure ERR defined by MUC [9][46] as shown in figure 8.

ERR removes the deweighting of D and I by simply removing the α weights in 7. The definition of ERR, however, still has the problem in that it implicitly deweights insertion errors relative to deletions and substitutions. This fact becomes more obvious when we rewrite figure 8 as [46]:

$$ERR = \frac{S + D + I}{N + I}$$

Figure 11: Rewritten error per response fill formula

ERR, the error measure defined by MUC, does have one aesthetic advantage in that it is guaranteed to be between 0 and 1, while SER, the slot error rate, can become greater than 1 under certain high error conditions [46].

3.3 Runtime performance measures

After discussing IE related performance measures, this chapter focuses on important factors which allow to measure the runtime performance of a program.

3.3.1 CPU time

An important metric to measure the runtime performance of a program is the process CPU time, which determines how much time a process spent in the CPU during its execution. CPU time is usually more comparable than pure execution time as it isn't influenced by other processes in a multitask environment. A lower CPU time means a program or an algorithm requires less time to execute, which is usually more preferable.

3.3.2 Memory consumption

But measuring the process CPU time is not enough. To compare the runtime performances of two different programs one needs to take the memory consumption into account, because different programs might use different time-memory tradeoffs. A lower memory footprint is usually more preferable, but trading CPU time for memory might lead to better results in certain scenarios. Assessing those memory tradeoffs usually differs from use case to use case. Memory consumption is usually measured in absolute dimension, e.g. Kilobytes, as opposed to a relative value based on the total memory size.

Memory consumption can be measured as an average value, but ocassionally the peak value might be of a certain interest as well.

3.4 Summary

This chapter presented and discussed the most widely used extraction and runtime performance values used to measure the overall system performance of extractors and how they relate to each other. Those values included performance measures, by name precision, recall, F-measure, error measure, error per response fill and slot error rate, as well as the runtime performance measures: CPU time and memory consumption. All of these performance measures will be integrated into the framework whose architecture, design and implementation details will be discussed in chapter 6 (see pp. 35ff.).

4 Modularity

Since the framework developed in the course of this thesis is required to be highly modular, we first need to define the term *Modularity*, find out how it relates to software engineering and choose a tool for supporting modularity on the Java platform.

Modularity is a frequently used term in Software Engineering. To understand the fundamental concept of it, take a look at the following definitions:

> Large software systems are inherently more complex to develop and maintain than smaller systems. Modularity involves breaking a large system into separate physical entities that ultimately makes the system easier to understand.
>
> **Java Application Architecture**
>
> Knoernschild [38]

> Systems are deemed "modular" when they can be decomposed into a number of components. The components are able to connect, interact, or exchange resources in some way, by adhering to a standardized interface. Unlike a tightly integrated product whereby each component is designed to work specifically with other particular components in a tightly coupled system, modular products are systems of components that are loosely coupled.
>
> **Modularity**
>
> Wikipedia [66]

Basically, modularity is based on modules, their requirements and behaviour. To fully understand the meaning of modularity we need to focus on the *module* itself:

4.1 Module definition

According to Knoernschild, a software module is defined as follows:

> A software module is a deployable, manageable, natively reusable, composable, stateless unit of software that provides a concise interface to consumers.

Java Application Architecture

Knoernschild [38]

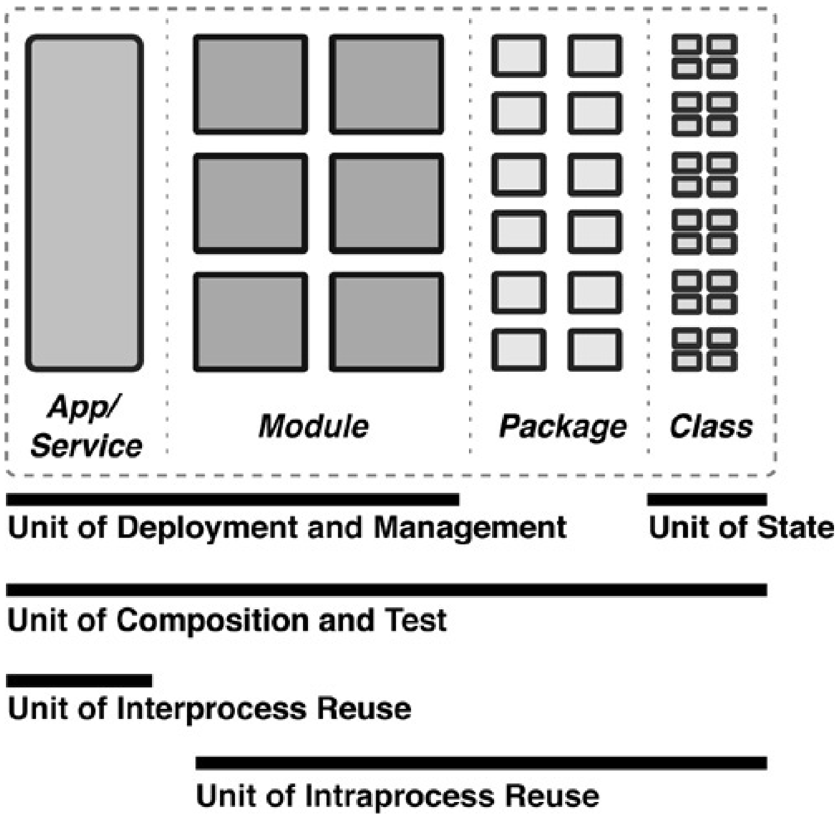

Figure 12: Module definition diagram

Figure 12 illustrates this defnition and all the individual aspects of a module [38]:

4.2 OSGi

OSGi is the most widely used and highly developed module system and service platform for the Java environment. This chapter aims to show why OSGi is the best choice for building highly modular Java-based software systems as it supports all requirements of modularity defined by Knoernschild[38].

The OSGi technology is a set of specifications that define a dynamic component system for Java. These specifications enable a development model where applications are dynamically composed of many different reusable components. The OSGi specifications enable modules to hide their implementations from other modules while communicating through services, which are objects that are specifically shared between modules. This surprisingly simple model has far reaching effects for almost any aspect of the software development process [50]. In OSGi parlance, a module is known as a bundle. OSGi provides a framework for managing bundles that are packaged as regular Java JAR files with an accompanying manifest. The manifest contains important metadata that describes the bundles and its dependencies to the OSGi framework [38]. Figure 13 shows the layered model architecture of the OSGi service platform.

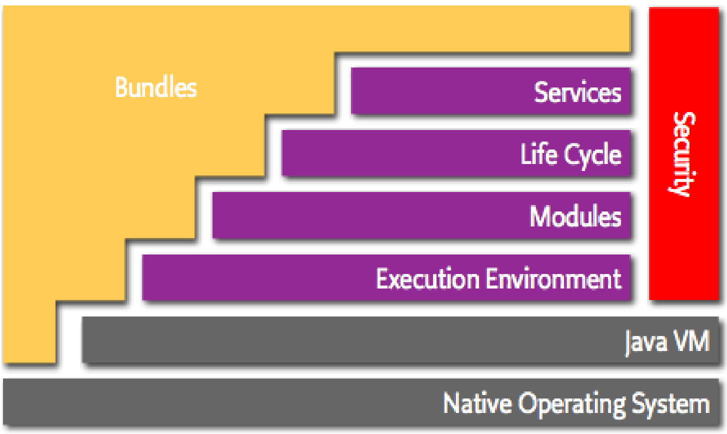

Figure 13: OSGi layered model [50]

4.2.1 Specification versions

The OSGi specification is under constant development and the most current version is *R5*, published in June 2012.

Name	Version	Date
OSGi Release 1	R1	May 2000
OSGi Release 2	R2	Octover 2001
OSGi Release 3	R3	March 2003
OSGi Release 4	R4	October 2005 / September 2006
OSGi Release 4.1	R4.1	May 2007
OSGi Release 4.2	R4.2	September 2009
OSGi Release 4.3	R4.3	April 2011
OSGi Release 5	R5	June 2012

Table 6: OSGi specification versions

Table 6 shows that OSGi is a proven, reliable platform specification, that has been under continuous development for many years.

4.2.2 Implementations

OSGi is the foundation for many different Application Servers and IDEs. Some of the most widely used open source implementations of the OSGi specification are listed here:

- **Eclipse Equinox**
  `http://eclipse.org/equinox/`
  Equinox is the core of the plug-in runtime for the Eclipse IDE.
- **Apache Felix**
  `http://felix.apache.org/`
  Apache Felix is the open source OSGi implementation powered by the Apache Software Foundation (ASF) and is the basis of several other Apache projects like Apache Aries and Apache Karaf.
- **Knopflerfish**
  `http://www.knopflerfish.org/`

> Knopflerfish is the spin-off from one of the OSGi alliance founding members and was open-sourced in 2003.

The developed framework uses Apache Felix for bundle testing purposes but aims to be OSGi compliant and implementation independence. Apache Felix was chosen for its easy configuration and small memory footprint.

4.3 Summary

OSGi is considered to be the most advanced module system for the Java platform. It supports all aspects of modularity, such as deployability, manageability, reusability and composability. Since modularity and its benefits and advantages, such as maintainability, is one of the main requirement of this work, OSGi is the best choice as a foundation for the framework.

5 Related Work

This chapter aims to provide a quick overview of some of the better known frameworks for evaluating information extraction systems.

5.1 Evaliex

Evaliex is an IE evaluation tool which integrates measurement concepts, like state-of-the-art scoring metrics, measuring string and semantic similarities and by parameterization of metric scoring, and provides an efficient user interface that supports evaluation control and the visualization of IE results.

To guarantee domain independence, the tool additionally provides a Generic Mapper for XML Instances (GeMap) which maps domain-dependent XML files containing IE results to generic ones. Compared to other tools, it provides more flexible testing and better visualization of extraction results for the comparison of different (versions of) information extraction systems [23].

Evaliex was part of a master thesis by Linsmayr in 2010: „Evaliex - Information Extraction Evaluation Framework“ [43]. A corresponding paper was published by Feilmayr, Pröll, and Linsmayr later in 2012: „EVALIEX - A Proposal for an Extended Evaluation Methodology for Information Extraction Systems“ [23].

Although Evaliex is a promising tool with a rich feature set; it's not available for other parties, as neither the tool itself nor the source code has yet been published. Evaliex also lacks a proper Application Programming Interfrace (API) as it is purely designed to be a standalone desktop application providing a Java-based user interface.

5.2 GATE

The General Architecture for Text Engineering (GATE)[1] is a Free & Open-Source infrastructure for developing and deploying software components that

[1] `http://gate.ac.uk/`

processes human language. It is more than 15 years old and in active use for all types of computational tasks involving language. GATE excels at text analysis of all shapes and sizes. [17].

The evaluation in GATE is provided by a component called the *Annotation-Diff Tool* which compares the individual annotations of a hypothesis with a reference. The differences are listed and visualized in color. GATE calculates the metrics recall, precision and F-measure [43].

Similar to Evaliex the GATE Annotation Diff Tool only supports a graphical user interface and is not designed to be embedded in or used by other systems. The tool also lacks error measures, by name error measure, error per response fill and slot error rate.

5.3 Ellogon

Ellogon[1] is a multi-lingual, cross-platform, general-purpose language engineering environment, developed in order to aid both researchers who are doing research in computational linguistics, as well as companies who produce and deliver language engineering systems. Ellogon, as a language engineering platform, offers an extensive set of facilities, including tools for processing and visualizing textual/HTML/XML data and associated linguistic information, support for lexical resources, tools for creating annotated corpora, accessing databases, comparing annotated data, or transforming linguistic information into vectors for use with various machine learning algorithms [21].

The deviation calculation of two collections of documents is provided by the *Collection Comparison tool*. It compares the annotations and attributes. After association it calculates precision, recall and F-measure [43].

The Ellogon Collection Comparison tool ships, very much like the Gate Annotation Diff Tool, only with a graphical user interface and lacks an API and the possibility to be embedded in other tools as a library. Ellogon also only supports the performance figures precision, recall and F-measure.

[1] `http://www.ellogon.org/`

5.4 ANNALIST

Annotation Alignment and Scoring Tool (ANNALIST)[1] is a scoring system for the evaluation of the output of semantic annotation systems. ANNALIST has been designed as a system that is easily extensible and configurable for different domains, data formats, and evaluation tasks. The system architecture enables data input via the use of plugins and the users can access the system's internal alignment and scoring mechanisms without the need to convert their data to a specified format. Although primarily developed for evaluation tasks that involve the scoring of entity mentions and relations, ANNALIST's generic object representation and the availability of a range of criteria for the comparison of annotations enables the system to be tailored to a variety of scoring jobs [18].

ANNALIST is, in contrast to the previously described systems, a pure evaluation tool. The data can be imported via special plug-ins and is processed by individual modules. The *Alignment Tool* associates hypotheses and references for each annotation type. The subsequent metric calculation is performed by the scoring module which determines precision, recall and the F-measure. The output module visualizes the results in a table [43].

ANNALIST meets the requirement of this thesis to offer an extensible evaluation tool by providing a plug-in mechanism. Plug-ins enable the tool to accept other input formats. In other words plug-ins translate between different input formats and required formats for the scoring module. ANNALIST comes with a Command Line Interface (CLI) and is therefore not developed with embeddability in mind.

5.5 Summary

All of these systems offer information extraction evaluation, some more sophisticated than others. Some of them are primary evaluation tools, others just support evaluation, next to several other features. But none of the tools described above is designed to be embedded in other software systems as a library. They usually only provide a user interface but not an API. But most

[1] `http://annalist.sourceforge.net/`

importantly, every tool only supports the evaluation of extraction results, not the runtime performance of different extractors.

Because none of these systems meets the requirements a new framework had to be designed which is based on the concepts of embeddability, extensibility, modularity and which combines the execution and evalutation of information extraction systems.

6 Design

This chapter presents the implemented framework *Banshie* (Benchmark Framework for Information Extraction). The requirements, architecture, design and concrete implementation details are explained and discussed on the following pages.

6.1 Analysis and requirements

The main goal was to provide a platform to benchmark domain-specific information extraction modules. Since the framework is planned to be used in a bigger platform and by other developers, it had to be designed for extension, modularity and embeddability. It should be based on the OSGi infrastructure and build with state-of-the-art patterns, like Dependency Injection (DI), Inversion of Control (IoC) and Composition over Inheritance in mind.

Since Banshie had to be developed in an Open Source fashion to support broader use cases, its whole code base as well as the documentation is available via

`https://github.com/whiskeysierra/banshie`

6.2 Architecture

Banshie is completely written in Java and distributed as OSGi bundles. The architecture of the framework was not the result of a Big Design Up Front (BDUF) but is rather based on a very rough design idea which allows to incrementally build in the design details as the project progresses. Knoernschild provides a catalog of architectural patterns for building highly modular systems in his book: *Java Application Architecture: Modularity Patterns with Examples Using OSGi (Robert C. Martin Series)* [38]

Almost all of Banshie's modules are the result of applying these patterns during the development. The following figure shows all relevant modules as well as their dependencies.

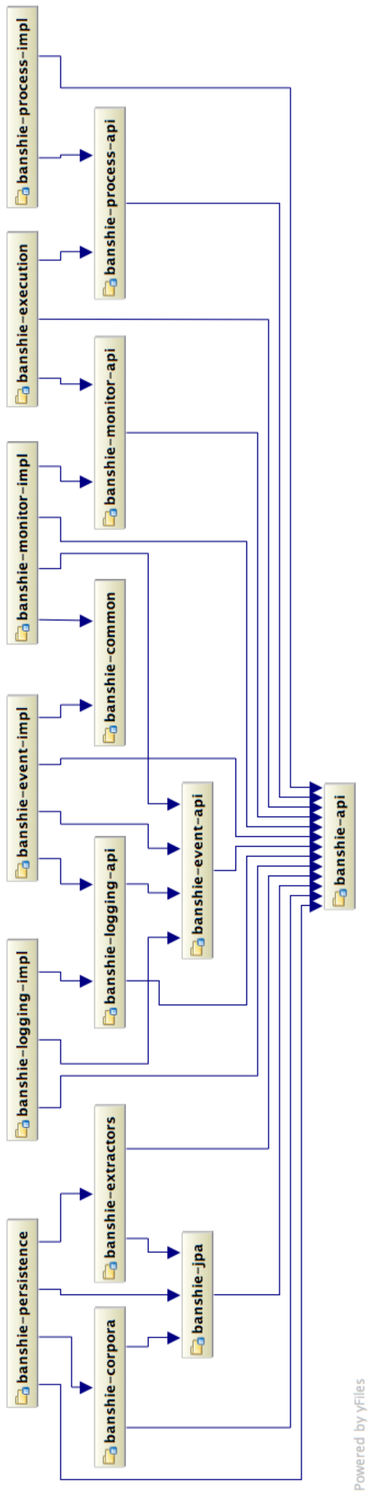

Figure 14: Module dependencies

6.3 Technologies and patterns

6.3.1 Dependency Injection

DI is an expression introduced by Martin Fowler in its article *Inversion of Control Containers and the Dependency Injection Pattern* [24]. Dependency Injection specifies the means for obtaining objects in such a way as to maximize reusability, testability and maintainability compared to traditional approaches such as constructors, factories, and service locators [37]. DI does this by allowing a class to specify its dependencies and rely on their provision at runtime rather than retrieving them explicitly. This leaves the programmer's code clean, flexible, and relatively free of dependency-related infrastructure [37].

Guice

Guice is a lightweight dependency injection framework for Java [29]. It's Open Source and available on `https://code.google.com/p/google-guice/`.

The typical code to implement Guice is shown in the following two listings. The first shows a simple *Module*. Modules in Guice are usually used to bind interfaces to concrete classes.

```
public final class ProcessModule extends AbstractModule {

    @Override
    protected void configure() {
        bind(ProcessService.class).to(DefaultProcessService.class);
    }

}
```

Listing 1: Guice module

In your classes you usually define a single constructor, annotated with `@Inject`, and all required dependencies as parameters. The construction of instances and the dependency resolution is done by Guice, no additional boilerplate code is necessary.

```java
final class DefaultEngine implements Engine {

    private final ProcessService service;

    @Inject
    DefaultEngine(ProcessService service) {
        this.service = service;
    }

}
```

Listing 2: Constructor injection

Guice Extensions

Guice has an extensible plug-in mechanism which allows third parties to provide additional functionality. Banshie uses two official Guice extension extensively: Assisted Inject[1] and Multibindings[2]. Assisted Inject allows the combination of Guice-provided dependencies and user-provided parameters on a single injection point. Multibindings supports the binding and injection of Sets and Maps.

Peaberry

Guice has no native OSGi support, apart from maybe the OSGi-compatible bundle manifest. To overcome this shortcoming, Peaberry[3], a third-party open-source Guice extension, offers OSGi-Guice bridge capabilities. It offers DI of OSGi dynamic services via Guice's common injection mechanisms and provides a rich and typesafe API to deal with the OSGi service registry and lifecycle events. Listing 3 shows the usage of Peaberry's lifecycle annotations.

[1] `https://code.google.com/p/google-guice/wiki/AssistedInject`
[2] `https://code.google.com/p/google-guice/wiki/Multibinding`
[3] `https://code.google.com/p/peaberry/`

```java
import org.ops4j.peaberry.activation.Start;

public class DefaultCorpusRepository implements CorpusRepository {

    private File basePath = new File("corpora");

    @Start
    public void onStart() {
        basePath.mkdirs();
    }

}
```

Listing 3: Peaberry lifecycle annotation

Peaberry even supports the automatic DI context creation upon bundle start by using an OSGi extender bundle. Bundles just need to provide the following bundle header to trigger an execution:

```
Bundle-Module:  org.whiskeysierra.banshie.execution.ExecutionModule
```

Listing 4: Peaberry bundle header

Peaberry creates one `Injector` per bundle, any interaction between bundles is based on standard OSGi services, which allows to combine Peaberry-aware bundles and normal ones.

6.3.2 Persistence

Banshie's persistence layer is based on the Java Persistence API (JPA) 2.0 Standard. JPA allows to build modules without hardcoding for a specific persistence provider or database vendor. Thus allowing to swap implementations later in the development lifecycle without the need to rewrite large portions of the code base. Banshie uses Apache OpenJPA as a JPA provider and Apache Derby as the underlying database. Derby is a Relational database management system (RDBMS) written in Java and is distributable as a single Jar file which can be embedded in other applications rather easily.

Using JPA in an OSGi environment is not a straight forward task. OSGi requires bundles to run in different and independent class loaders, while JPA heavily relies on classpath scanning and reflection. Both techniques don't work quite well together. Because JPA-based persistence is a common requirement, the *OSGi Service Platform Release 4 Version 4.2 Enterprise Specification* addressed this issue and specifies a standard way to define *Persistence* and *Client bundles* [51]. A persistence bundle is a bundle with the following bundle header:

```
Meta-Persistence:  META-INF/persistence.xml
```

Listing 5: Persistence bundle header

A client bundle is just a bundle that makes use of the `EntityManagerFactory` provided by the corresponding persistence unit. Most OSGi containers delegate this part of the OSGi specification to third-party libraries and bundles. Apache Aries aims to provide portable implementations in the form of standard OSGi bundles for those parts of the OSGi specification. Banshie uses the JPA module of Apache Aries consisting of the *Aries JPA API bundle* and the *Aries JPA container bundle*. To minimize common boilerplate code and manual transaction handling, all JPA client bundles use the Guice extension *Guice Persist*. Guice Persist offers AOP-interception for annotated methods as shown in the following listing:

```
class DefaultCorpusRepository implements CorpusRepository {

    private EntityManager manager() {
        return provider.get();
    }

    @Transactional
    @Override
    public Corpus get(UUID uuid) {
        return manager().find(CorpusEntity.class, uuid);
    }

}
```

Listing 6: Guice Persist annotation

6.3.3 Build tools

BND Tool and the Maven Bundle Plugin

With OSGi you are forced to provide additional metadata in the JAR's manifest to verify the consistency of your classpath. This metadata must be closely aligned with the class files in the bundle. Maintaining this metdata is an error prone chore because many aspects are redundant. The core task of the BND Tool is to analyze the class files and find every dependency. These dependencies are then merged with instructions supplied by the user [5]. Since Banshie uses Apache Maven for building its independent modules, the natural choice was to use a Maven Plugin for this, which is provided by the Apache Felix Maven Bundle Plugin [1]. The following listing shows the bare minimum of configuration code to use the Maven Bundle Plugin in a POM file.

```
<packaging>bundle</packaging>
...
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.felix</groupId>
            <artifactId>maven-bundle-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```

Listing 7: Maven Bundle Plugin usage

[1] http://felix.apache.org/site/apache-felix-maven-bundle-plugin-bnd.html

6.4 API

The framework's API can be divided into three main components, which are described in detail on the following pages.

Domain model and persistence

Banshie's domain model has been designed with simplicity and extensibility in mind. The two only entity classes, `Corpus` and `Extractor`, are merely containers for file locations and very little meta data, but since the persistence layer is based on JPA, which is based on *Plain Old Java Objects*, adding properties is a rather easy task.

An extractor holds the name, version and the path to the executable jar file, while a corpus identifies the reference output as well as a related input document.

For each of the model classes a persistence service interface is provided. Since the framework in it current stage does not require very sophisticated features, the interface of these services is intentionally kept to a minimum.

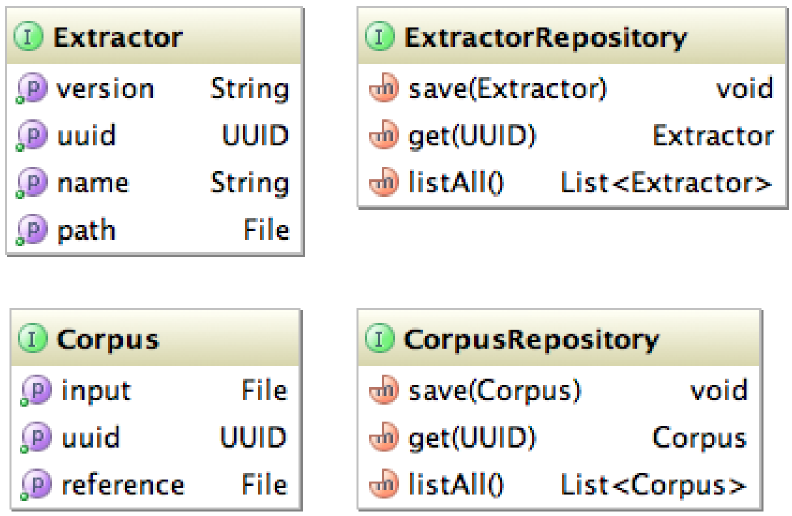

Figure 15: Banshie model and persistence API

Execution

Apart from the domain model and persistence layer, the framework offers two significant features to its clients: execution and evaluation. The execution package contains one major interface for executing extractors: the `Engine`. The Engine provides a single methods which takes an Extractor-Corpus-pair and performs an execution. The result of the extractor run is then passed back to the caller in the form of an `ExtractorResult` containing references to the extractor's xml output file and the event log file. For more details about the structure of the event log file please consult chapter 6.7.

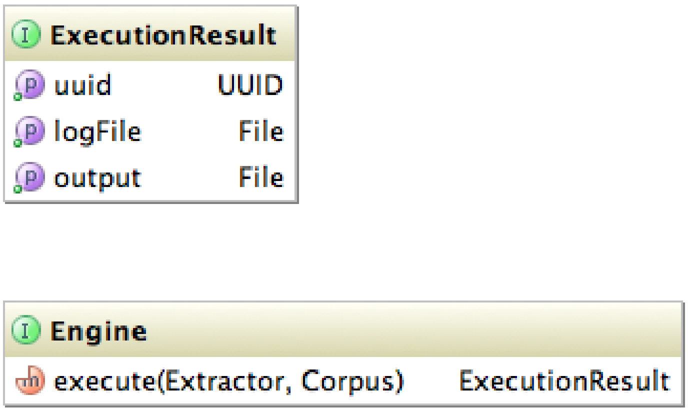

Figure 16: Banshie Execution API

Evaluation

Evaluation is, next to the execution, the other main feature of the framework. Having a two-step phase for execution and evaluation has an advantage. Persisting the extraction results and raw performance logging data allows for a later re-evaluation by other versions or differently configured extractors. Additionally, saving results on the filesystem allows for more memory efficiency.

The evaluation packages contains several type definitions as shown in figure 17:

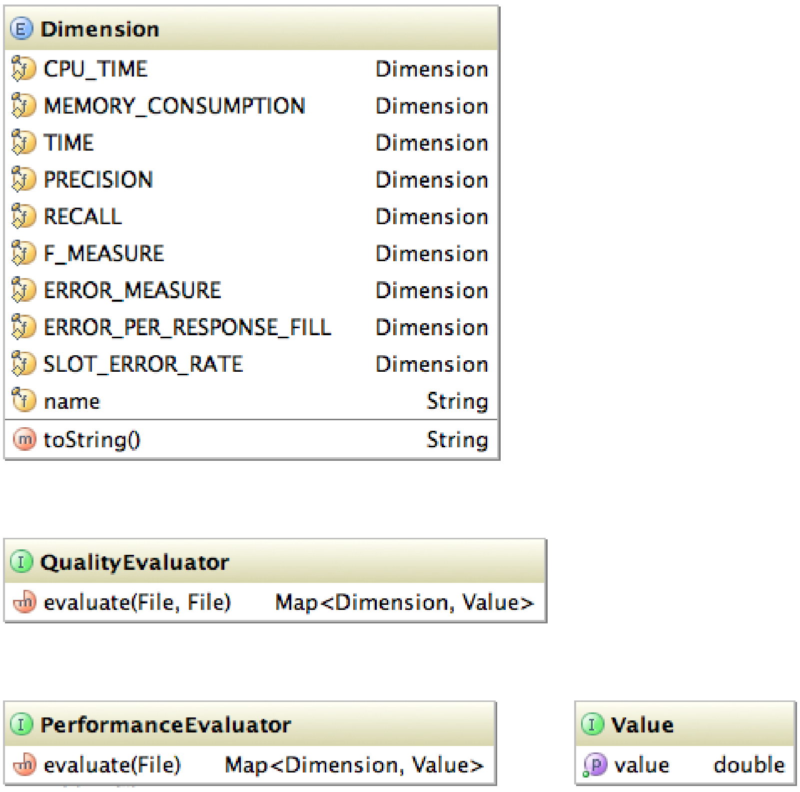

Figure 17: Banshie Evaluation API

The two main service types are `QualityEvaluator` and `PerformanceEvaluator`. Both have a very similar interface, since they provide a single method to evaluate the execution result. The QualityEvaluator compares the Corpus' reference and the Extractor's hypothesis and calculates quality performance measures like Precision, Recall and F-Measure. The PerformanceEvaluator on the other hand focuses on calculating runtime performance measures, like CPU time, memory consumption and execution time by processing the event log file.

API Usage

Listing 8 shows the simple basic steps required to perform a single extractor execution and evaluation.

```java
// via dependency injection or direct instantiation
final ExtractorRepository extractors = ...;
final CorpusRepository corpora = ...;
final Engine engine = ...;
final PerformanceEvaluator performance = ...;
final QualityEvaluator quality = ...;

final Extractor extractor = extractors.get(extractorId);
final Corpus corpus = corpora.get(corpusId);

final ExecutionResult result = engine.execute(extractor, corpus);
final Map<Dimension, Value> p =
    performance.evaluate(result.getLogFile());
final Map<Dimension, Value> q =
    quality.evaluate(corpus.getReference(), result.getOutput());

// handle evaluation results
...
```

Listing 8: Banshie API usage

6.5 Extractor interface specification

Since Banshie aims to evaluate the extraction quality as well as the runtime performance of information extraction systems, it sets some special requirements for extractors.

Any extractor evaluated by the framework is required to be written in Java and compiled as a single executable Jar file for Java 1.6 or higher. Being packaged as a single file requires the extractor to bundle every external dependency into a single archive. Embedding third-party java libraries can be accomplished by utilizing the *JarJar*[1] tool. External files, like models and training data, can be packaged as standard classpath resources.

For a Jar file to be executable it has to have a manifest file, i.e. META-INF/MANIFEST.MF), and a manifest header as shown in the following listing:

```
Main-Class:  org.whiskeysierra.banshie.example.opennlp.Main
```

Listing 9: Extractor manifest header

The extractor can then be started using the following command:
`java -jar extractor.jar`

Since an extractor under evaluation has a single input, the test document, and a single output, the annotated hypothesis, the natural choice was to utilize standard streams, standard input (*stdin*) and standard output (*stdout*) respectively. The test document is passed to the extractor as plain text in UTF-8 encoding. Whether the extractor streams the document or reads it into memory as a whole is up to the extractor. The output format is Extensible Markup Language (XML) as defined by the schema shown in listing 10.

It should be explicitly stated, that this approach, in its current, form only supports single document extraction.

[1] `https://code.google.com/p/jarjar/`

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="document">
    <xs:complexType mixed="true">
      <xs:sequence>
        <xs:element name="span" maxOccurs="unbounded">
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:string">
                <xs:attribute name="type" type="xs:string"/>
                <xs:attribute name="start" type="xs:int"/>
                <xs:attribute name="end" type="xs:int"/>
              </xs:extension>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Listing 10: Banshie XML Schema

As shown in listing 11, the defined output format is a very simple XML document containing the original document and all found entities annotated with the corresponding type as a simple XML element tag.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<document xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation="schema.xsd">
    <span type="person" start="0" end="14">Albert Einstein</span>
    (14 March 1879 - 18 April 1955) was a German-born theoretical
    physicist who developed the general theory of relativity,
    effecting a revolution in physics. For this achievement,
    <span type="person" start="176" end="184">Einstein</span> is
    often regarded as the father of modern physics and the most
    influential physicist of the 20th century. While best known
    for his mass-energy equivalence formula E = mc2 (which has been
    dubbed "the world's most famous equation"), he received the 1921
    Nobel Prize in Physics "for his services to theoretical physics,
    and especially for his discovery of the law of the photoelectric
    effect". The latter was pivotal in establishing quantum theory.
</document>
```

Listing 11: Banshie XML Example

The span element has three attributes: `type`, `start` and `end`. Type is one of `person`, `organization`, `date` or `location`, based on the ENAMEX tags developed for the Message Understanding Conference [33].

The attributes `start` and `end` define the UTF-8 character offset of the span in the original document to support character based association of spans in the reference and the predication.

6.6 Reference-hypothesis association

Douthat proposed the original version of the „General Greedy Mapping Algorithm“ in 1998 [19]. It's based on finding matching pairs of spans in the reference and the predication. Evaliex extended this algorithm by basing the matching on string or word similarity algorithms like Levenshtein-distance or the Jaccard-coeffecient [43]. But since Banshie, in its current version, focuses solely on Named Entity Recognition, a simpler algorithm to associate reference and hypothesis has been used. Pairs are matched based on character offsets calculated from the original document. This way an extractor is required to find exact or partial matches of spans defined in the reference to score in the evaluation metrics.

6.7 Implementation

6.7.1 Engine

The *Engine* interface offers a single facade to executing an `Extractor` against a supplied `Corpus`. It's the Engine's responsibility to provide an independent and isolated execution environment for extractors, to manage process creation and lifecycle and collect and persist runtime events.

After an initial design draft, it was clear, that the Engine implementation will be too big for a single module. So the engine implementation was split up into multiple smaller, more maintainable modules using the module development patterns, defined by Knoernschild[38] and discussed in chapter 4. The engine's submodules are described in the following chapters.

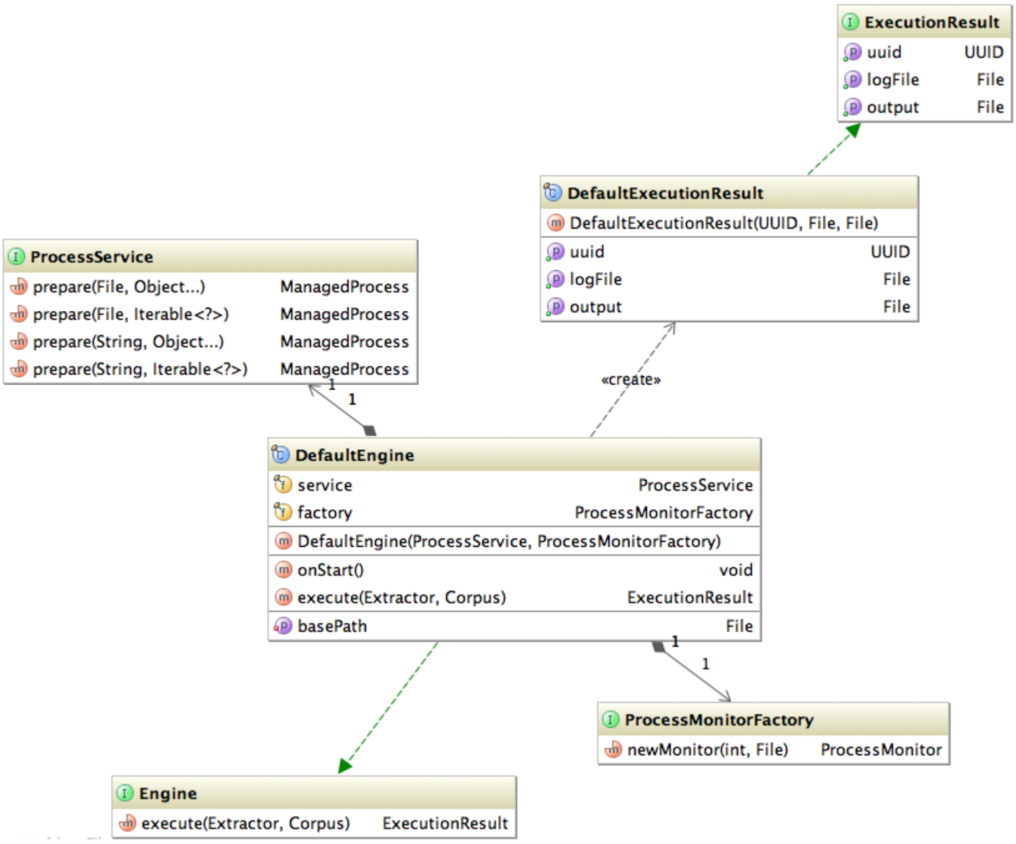

Figure 18: Engine implementation

Process management

The Java Process API is part of the Java Runtime Environment since version 1.0 but it has several pitfalls, which may lead to deadlocks or zombie processes [7]. There even has been filed a JDK Enhancement-Proposal (JEP) to improve the API for controlling and managing operating system processes. [4].

Since executing an extractor in an isolated, independent execution environment is a crucial part of the Engine's task the framework required a cleaner, more fail-safe API for process creation and lifecycle management that integrates nicely with the rest of the framework and it's main general purpose library, i.e. Guava. Listing 12 demonstrates the minimal steps to use the framework's Process API while figure 19 shows the relevant parts of it.

```
final ManagedProcess managed = service.prepare("java", "-version");
final RunningProcess process = managed.call();

// write to process.getOutput() or
// read from process.getInput()

process.await();
// or process.cancel()
```

Listing 12: ProcessService API example usage

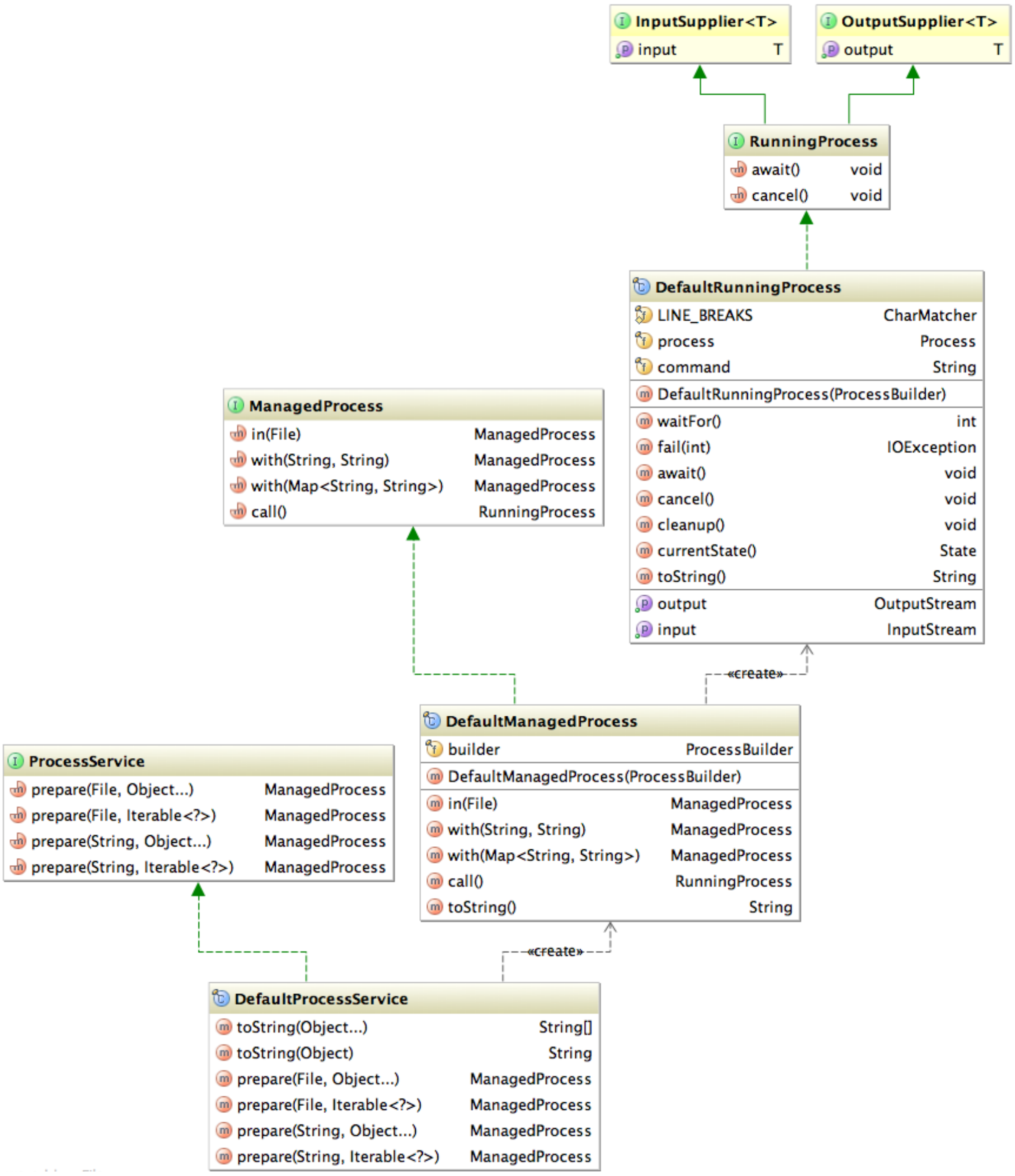

Figure 19: ProcessService implementation

Process monitoring

After an extractor has been started in its own environment using the ProcessService API, the operating system process needs to be monitored to ensure correct execution and to collect events, like current CPU time and memory consumption, in a periodical way, e.g. every x milliseconds.

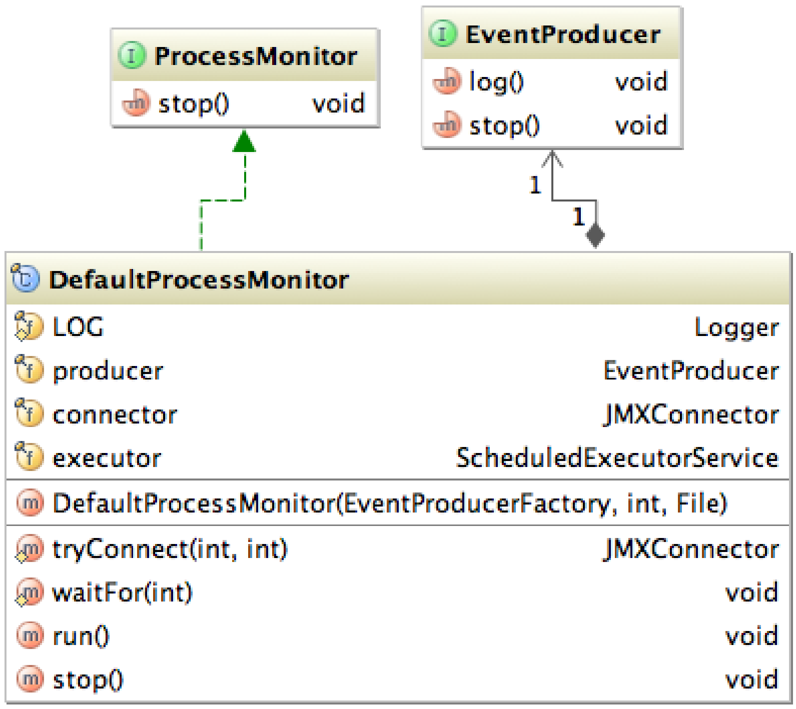

Figure 20: ProcessMonitor implementation

The repeated polling is realized with the JDK's built-in `ScheduledExecutorService`[1], which schedules a special `Runnable` to run in a scheduled fashion, like shown in listing 13

```
executor.scheduleAtFixedRate(runnable, 0L, 1L, TimeUnit.SECONDS);
```

Listing 13: Scheduling in java

[1]`http://docs.oracle.com/javase/6/docs/api/java/util/concurrent/ScheduledExecutorService.html`

Collecting events on a separate process can be realized by using Java Management Extensions (JMX), which allows managing and monitoring java applications through a well specified and extensible interface. To allow a JMX connection, a java process needs to be started with a special set of command line parameters as shown in listing 14.

```
final ManagedProcess managed = service.prepare(
    "java",
    "-Dcom.sun.management.jmxremote",
    "-Dcom.sun.management.jmxremote.port=" + port,
    "-Dcom.sun.management.jmxremote.authenticate=false",
    "-Dcom.sun.management.jmxremote.ssl=false",
    "-jar", extractor.getPath()
);
```

Listing 14: Configuring process to use JMX

After the process has been started, a JMX connection can be established with the following steps:

```
final String url = "service:jmx:rmi:///jndi/rmi://localhost:" +
    port + "/jmxrmi";
final JMXServiceURL serviceUrl = new JMXServiceURL(url);
final JMXConnector connector =
    JMXConnectorFactory.connect(serviceUrl, null);
```

Listing 15: JMX connection

The default implementation of the `ProcessMonitor` interface delegates the work of creating events to another service, the `EventProducer`, by passing on the `JMXConnector`.

It's worth mentioning that relying on JMX for measuring performance values of an external extraction process is the reason why Banshie is currently limited to JVM-based extractors.

Event production

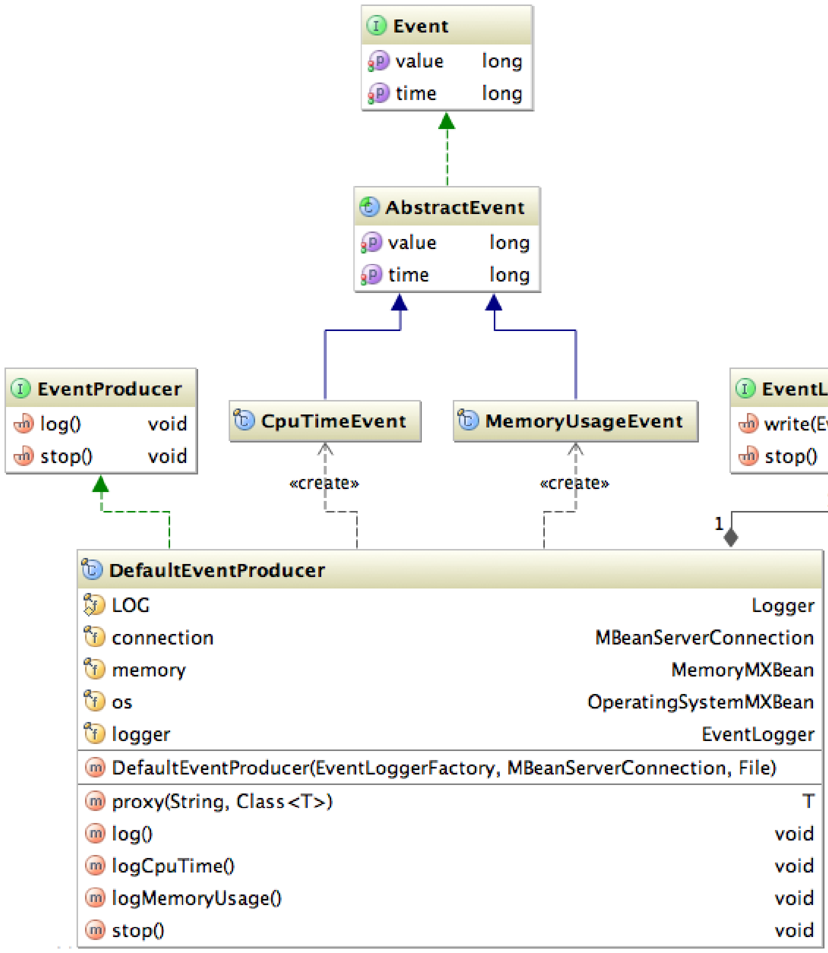

Figure 21: EventProducer implementation

It's the `EventProducer`'s responsibility to retrieve runtime performance figures by calling the corresponding JMX endpoints and to create and populate the suitable `Event` instances. The producer generates JMX proxies for the `MemoryMXBean`[1] and `OperatingSystemMXBean`[2] classes, creates

[1]`http://docs.oracle.com/javase/6/docs/api/java/lang/management/MemoryMXBean.html`

[2]`http://docs.oracle.com/javase/6/docs/jre/api/management/extension/com/sun/management/OperatingSystemMXBean.html`

`CpuTimeEvent`s and `MemoryUsageEvent`s and delegates their persistence to the `EventLogger` service.

Event persistence

Once events have been created, they need to be persisted on the file system to allow processing them during evaluation later on. Event persistence in Banshie is provided by the default `EventLogger` implementation, which creates a single text-oriented log file using the JavaScript Object Notation (JSON) data format. `Event`s are serialized using JSON mapping capabilities of the Jackson[1] library. The resulting event log file contains one JSON entity per line.

Using JPA for persisting events would have certainly been an option, but using a text-based logfile approach has its advantages. First of all, it's more resource friendly to stream text to a file than it is to go through several layers of abstraction to access a database via JPA. Another reason to use JSON for serializing is the flexibility it provides; one could easily add more information to events in the future by just extending the JSON structure with additional key-value pairs.

[1] `http://jackson.codehaus.org/`

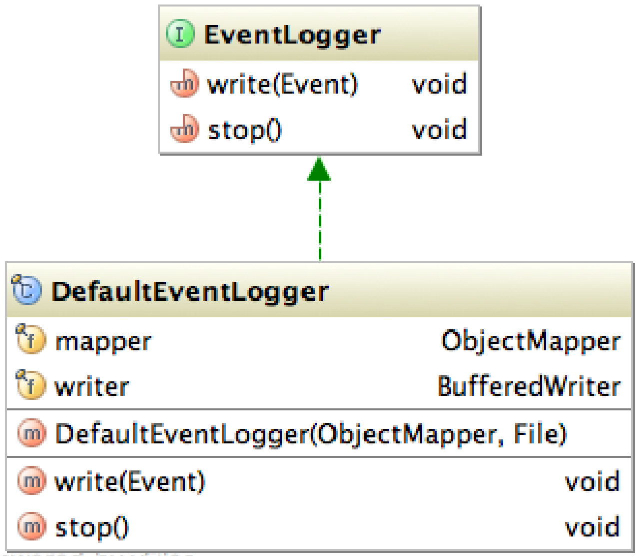

Figure 22: EventLogger implementation

```
{"type":"cpu","time":1362932786442,"value":630000000}
{"type":"memory","time":1362932786870,"value":21402296}
{"type":"cpu","time":1362932787374,"value":1160000000}
{"type":"memory","time":1362932787399,"value":34213984}
{"type":"cpu","time":1362932788375,"value":2620000000}
{"type":"memory","time":1362932788423,"value":90823392}
{"type":"cpu","time":1362932789375,"value":3790000000}
{"type":"memory","time":1362932789383,"value":162860136}
{"type":"cpu","time":1362932790375,"value":6810000000}
{"type":"memory","time":1362932791603,"value":239290480}
{"type":"cpu","time":1362932791612,"value":6880000000}
{"type":"memory","time":1362932791614,"value":240990136}
{"type":"cpu","time":1362932792374,"value":7840000000}
{"type":"memory","time":1362932792376,"value":304883808}
{"type":"cpu","time":1362932793375,"value":9110000000}
{"type":"memory","time":1362932793378,"value":313170504}
```

Listing 16: Example event log file excerpt

6.7.2 Quality evaluation

The default `QualityEvaluator` implementation has serveral tasks. It needs to map the prediction to the reference, count true positives, false positives, false negatives, run configured scoring metrics and collect their results.

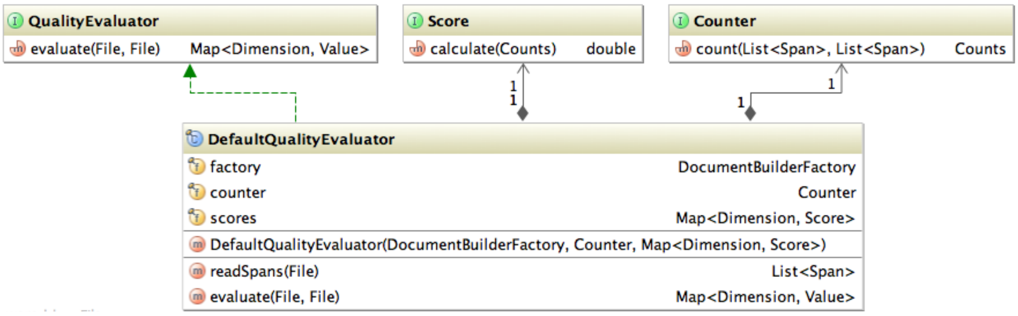

Figure 23: QualityEvaluator implementation

The reference-hypothesis association is provided by the `Counter` class, as shown in figure 24. For details about the reference-hypothesis association algorithm used, please consult chapter 6.6.

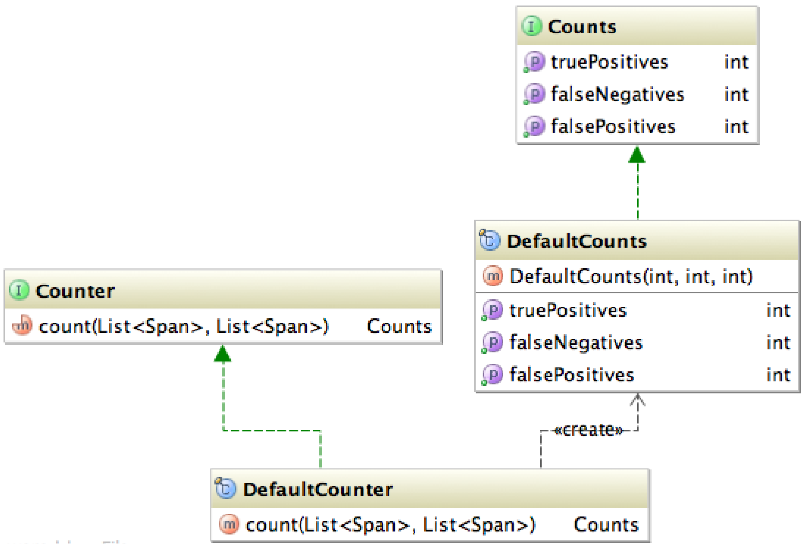

Figure 24: Counter implementation

The scoring metrics are realized by implementations of the `Score` interface. Every performance metric discussed in chapter 3 has been implemented as single `Score` implementation.

Metric	Implementation class
Precision (ρ)	`Precision`
Recall (π)	`Recall`
F-measure (F)	`FMeasure`
Error measure (E)	`ErrorMeasure`
Error per response fill (ERR)	`ErrorPerResponseFill`
Slot error rate (SER)	`SlotErrorRate`

Table 7: Metric implementations

Since several metrics are based on others, implementations are free to reuse instances of other types as shown in the following package diagram.

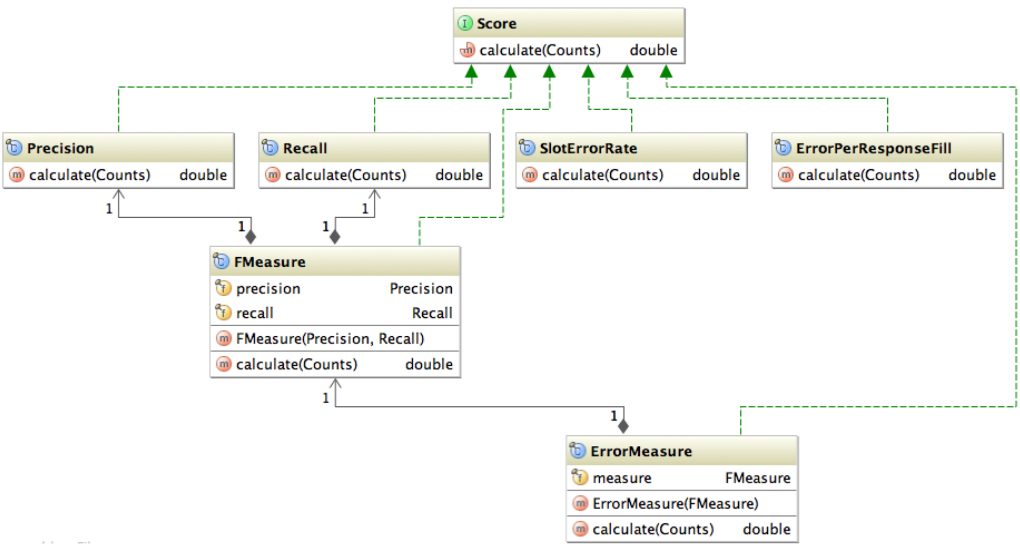

Figure 25: Score implementation

Listing 17 shows an example `Score` implementation, in this case `Recall`.

```java
final class Recall implements Score {

    @Override
    public double calculate(Counts counts) {
        final double sum = counts.getTruePositives() +
            counts.getFalseNegatives();

        if (sum > 0) {
            return counts.getTruePositives() / sum;
        } else {
            // cannot divide by zero, return error code
            return Double.NaN;
        }
    }

}
```

Listing 17: Recall score implementation

6.7.3 Performance evaluation

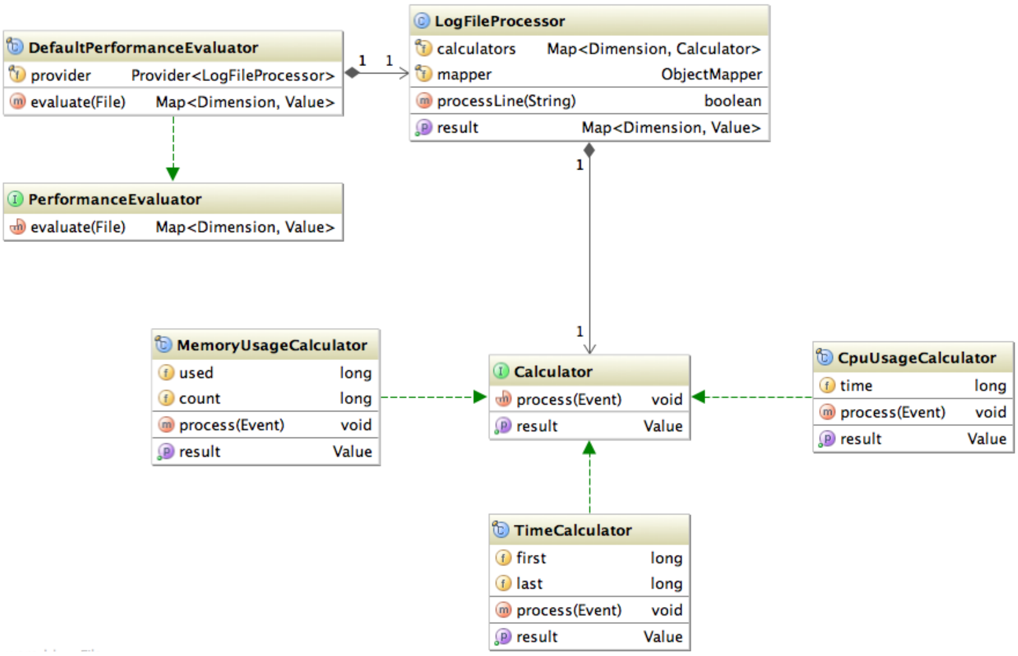

Figure 26: PerformanceEvaluator implementation

Runtime performance evaluation is a little bit easier than quality evaluation. The default `PerformanceEvaluator` implementation needs to read the event log file line by line, deserialize events, update all configured calculators and finally collect their results.

```
@Override
public boolean processLine(String line) throws IOException {
    final Event event = mapper.readValue(line, Event.class);

    for (Calculator calculator : calculators.values()) {
        calculator.process(event);
    }

    return true;
}
```

Listing 18: LogFileProcessor

A `Calculator` is similar to the `Score` interface shown earlier, except that calculators are inherently stateful.

```
interface Calculator {

    void process(Event event);

    Value getResult();

}
```

Listing 19: Calculator Interface

Banshie, in its current version, ships with three different `Calculators`:

- `CpuUsageCalculator`
  Retrieves the CPU time from a stream of events.
- `MemoryUsageCalculator`
  Calculates the average memory consumption in megabytes.
- `TimeCalculator`
  Calculates the total execution time in milliseconds.

Listing 20 demonstrates a common `Calculator` implementation at the example of the `MemoryUsageCalculator` which computes the average memory consumption in megabytes.

```
final class MemoryUsageCalculator implements Calculator {

    private long used;
    private long count;

    @Override
    public void process(Event e) {
        if (e instanceof MemoryUsageEvent) {
            final MemoryUsageEvent event =
                MemoryUsageEvent.class.cast(e);

            used += event.getValue() / 1024L / 1024L;
            count++;
        }
    }

    @Override
    public Value getResult() {
        return new SimpleValue(used / count);
    }

}
```

Listing 20: MemoryUsageCalculator

6.8 Summary

This chapter described the development of Banshie, an extensible and modular evaluation framework for information extraction systems and also showed how the findings from chapter 3 (Formal IE evaluation methodology) and 4 (Modularity) influenced the overall system design and fundamental architecture of the framework. Various design and implementation details have been discussed, such as the API, the extractor interface specification, the output format definition as well as the reference-hypothesis association algorithm and all relevant modules, including their respective requirements and interfaces.

7 Results

This chapter aims to show the results of the framework implementation by demonstrating how existing information extraction systems can be evaluated with Banshie. For evaluation purposes two freely available NER taggers were chosen. A test document, reference output and examplary adapter implementations for the selected systems will be shown and discussed over the course of the following pages.

7.1 Open Source Extractors

Several NLP tools and suites exist but for the purpose of evaluating Banshie only two were chosen. The choice came down to Apache OpenNLP and Stanford CoreNLP because both are well-known tools from respected authorities in Open Source Software and Natural Language Processing: in this case the ASF and Christopher Manning[1] respectively. A second reason for selecting those two is that both are written in Java and available via the Maven Central repository.

7.1.1 Apache OpenNLP

Apache OpenNLP[2] is a machine learning based toolkit for the processing of natural language text. It supports the most common NLP tasks, such as tokenization, sentence segmentation, part-of-speech tagging, named entity extraction, chunking, parsing, and coreference resolution. [2]

The Apache OpenNLP project is written in Java, licensed under the Apache License Version 2.0, available on Maven Central and completed incubation on February 2012 and is now an Apache top level project.

[1] `http://nlp.stanford.edu/~manning/`
[2] `http://opennlp.apache.org/`

7.1.2 Stanford CoreNLP

Stanford CoreNLP [1] is an integrated framework, which make it very easy to apply a bunch of language analysis tools to a piece of text. Stanford CoreNLP integrates tools, like the a part-of-speech tagger, a named entity recognizer, a parser, and a coreference resolution system, and provides model files for analysis of English. The goal of the project is to enable people to quickly and painlessly get complete linguistic annotations of natural language texts. It is designed to be highly flexible and extensible. [62]

The Stanford CoreNLP code is written in Java, licensed under the GNU General Public License and available on Maven Central.

[1] `http://nlp.stanford.edu/software/corenlp.shtml`

7.2 Examples

This chapter will show the most relevant parts of the adapter implementations to make the selected NER tools compatible to the extractor interface specification described in chapter 6.5. For the sake of readability of this document and to keep the evaluation process understandable an extremely small test document, shown in listing 21, was chosen.

```
Pierre Vinken, 61 years old, will join the board as a
nonexecutive director Nov.  29.  Mr.  Vinken is chairman of
Elsevier N.V., the Dutch publishing group.  Rudolph Agnew, 55
years old and former chairman of Consolidated Gold Fields PLC,
was named a director of this British industrial conglomerate.
```

Listing 21: Example document

The example reference output, listing 22, was created manually by annotating the example document with xml tags. It should be noted, that an editor visualizing the UTF-8 character offset is extremely helpful when creating a reference output by hand. Another possibility is to take the output of one extractor and modify the result accordingly.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<document>
    <span type="person" start="0" end="13">Pierre Vinken</span>,
    61 years old, will join the board as a nonexecutive director
    <span type="date" start="76" end="83">Nov. 29</span>.
    Mr. <span type="person" start="89" end="95">Vinken</span>
    is chairman of <span type="organization" start="111"
    end="124">Elsevier N.V.</span>, the Dutch publishing group.
    <span type="person" start="154" end="167">Rudolph Agnew
    </span>, 55 years old and former chairman of
    <span type="organization" start="205" end="233">Consolidated
    Gold Fields PLC</span>, was named a director of this
    British industrial conglomerate .
</document>
```

Listing 22: Example extraction reference

7.2.1 Apache OpenNLP

Listing 23 shows the most relevant part of the adapter implementation for the Apache OpenNLP Name Finder Tool, with omitting the document reading and output writing part of the code. The three most important services provided by OpenNLP are the `SentenceDetector`, the `Tokenizer` and the `NameFinder`. The SentenceDetector and Tokenizer are used to split up the input document into sentences and sentences into tokens respectively. The resulting lists of tokens are than passed on to the NameFinder which finds spans and marks them as persons, organizations or locations accordingly.

```java
final SentenceDetector detector = getSentenceDetector();
final Tokenizer tokenizer = getTokenizer();
final TokenNameFinder finder = getNameFinder();

for (Span sentences : detector.sentPosDetect(document)) {
    final String sentence = sentences.getCoveredText(document);
    final Span[] indices = tokenizer.tokenizePos(sentence);
    final String[] tokens = spansToStrings(indices, sentence);
    final Span[] spans = finder.find(tokens);
    final List<String> words = Arrays.asList(tokens);

    for (Span span : spans) {
        final String word = joiner.join(
            words.subList(span.getStart(), span.getEnd()));
        final String type = span.getType();
        final int start = sentences.getStart() +
            indices[span.getStart()].getStart();
        final int end = sentences.getStart() +
            indices[span.getEnd()].getEnd() - 1;

        // ...
    }
}
```

Listing 23: Apache OpenNLP extractor adapter

OpenNLP's NER is primarily designed to work with sentences, but since our schema requires character offsets, some additional index counting is requiring.

Listing 24 shows the extraction result produced by the adapter implementation.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<document>
    <span type="person" start="0" end="13">Pierre Vinken</span>,
    61 years old , will join the board as a nonexecutive
    director Nov. 29 .Mr. Vinken is chairman of Elsevier N.V.,
    the Dutch publishing group. <span type="person"
    start="154" end="167">Rudolph Agnew</span>, 55 years old
    and former chairman of Consolidated Gold Fields PLC , was
    named a director of this British industrial conglomerate.
</document>
```

Listing 24: Apache OpenNLP extraction result

The evaluation result of the extraction outcome and the recorded event log produced by the default `PerformanceEvaluator` and `QualityEvaluator` implementations, discussed in chapter 6, is shown in following listing.

```
CPU_TIME: 2250.0
MEMORY_CONSUMPTION: 35.0
TIME: 2001.0
PRECISION: 1.0
RECALL: 0.3333333333333333
F_MEASURE: 0.5
ERROR_MEASURE: 0.5
ERROR_PER_RESPONSE_FILL: 0.6666666666666666
SLOT_ERROR_RATE: 0.6666666666666666
```

Listing 25: Apache OpenNLP evaluation result

The discrepancy between CPU time and execution is based on the fact that the process CPU time provided by `OperatingSystemMXBean` is of nanoseconds precision but not necessarily nanoseconds accuracy[34] while the system time is in milliseconds. During extremely short startup times, like in this case, it might cause the values to inaccurate.

As shown in listing 25, OpenNLP only required about 2.25 seconds CPU time and 35 megabytes of memory to perform the extraction but on the other it only scored a recall of .3 and a F-measure of .5, which means the OpenNLP tagger only found a third of what could have been found, according to our reference output.

7.2.2 Stanford CoreNLP

The API provided by Stanford CoreNLP is different from Apache OpenNLP as it is designed a pipelines. It's possible to add different annotators to a pipeline which work on any document passed down the pipeline. The most common annotators are `tokenize` (tokenizing), `ssplit` (sentence splitting), `pos` (part-of-speech tagging) and `ner` (named entity recognition). The results produced by the annotators can then be retrieved by using the corresponding `get`-methods on sentence or token instances as shown in listing 26.

```
final Annotation annotation = new Annotation(document);

final Properties properties = new Properties();
properties.put("annotators", "tokenize, ssplit, pos, lemma, ner");
final StanfordCoreNLP pipeline = new StanfordCoreNLP(properties);

pipeline.annotate(annotation);

final List<CoreMap> sentences =
    annotation.get(SentencesAnnotation.class);

for (CoreMap sentence : sentences) {
    for (CoreLabel token : sentence.get(TokensAnnotation.class)) {
        final String word =
            token.get(TextAnnotation.class);
        final String type =
            token.get(NamedEntityTagAnnotation.class);
        final int start = token.beginPosition();
        final int end = token.endPosition();

        // ...
    }
}
```

Listing 26: Stanford CoreNLP extractor adapter

The extraction of the CoreNLP adapter is shown in listing 27 and it's obvious that it contains much more annotated tokens than the OpenNLP result in listing 24. But since the Stanford NER tagger works on tokens, it tags individual words with NER types.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<document>
    <span type="person" start="0" end="6">Pierre</span>
    <span type="person" start="7" end="13">Vinken</span>,
    61 years old, will join the board as a nonexecutive director
    <span type="date" start="76" end="80">Nov.</span>
    <span type="date" start="81" end="83">29</span>.
    Mr. <span type="person" start="89" end="95">Vinken
    </span> is chairman of
    <span type="organization" start="111" end="119">Elsevier</span>
    <span type="organization" start="120" end="124">N.V.</span>,
    the Dutch publishing group.
    <span type="person" start="154" end="161">Rudolph</span>
    <span type="person" start="162" end="167">Agnew</span>,
    55 years old and former chairman of
    <span type="organization" start="205" end="217">Consolidated</span>
    <span type="organization" start="218" end="222">Gold</span>
    <span type="organization" start="223" end="229">Fields</span>
    <span type="organization" start="230" end="233">PLC</span>,
    was named a director of this British industrial conglomerate.
</document>
```

Listing 27: Stanford CoreNLP extraction result

Listing 28 shows the evaluation result of the CoreNLP adapter.

```
CPU_TIME: 40990.0
MEMORY_CONSUMPTION: 575.0
TIME: 44934.0
PRECISION: 1.0
RECALL: 1.0
F_MEASURE: 1.0
ERROR_MEASURE: 0.0
ERROR_PER_RESPONSE_FILL: 0.0
SLOT_ERROR_RATE: 0.0
```

Listing 28: Stanford CoreNLP evaluation result

It quickly becomes apparent, that Stanford CoreNLP produces perfect extraction results, according to our example reference, but does so on the expense of required CPU time and memory consumption.

Name	t_{cpu}	m	t	π	ρ	F	E	ERR	SE
OpenNLP	2250.0	35.0	2001.0	1.0	0.33	0.5	0.5	0.66	0.66
CoreNLP	40990.0	575.0	44943.0	1.0	1.0	1.0	0.0	0.0	0.0

Table 8: Evaluation result comparison

Table 8 shows the evaluation results of both extractors in directo comparison.

CoreNLP actually spends a lot of time loading training models, but since OpenNLP also loads model data during startup, comparing the total execution time is still fair. But it should be noted, that on consecutive run, CoreNLP should perform much better. One could even imagine to use different models to trade startup time for extraction quality. Which is exactly the kind of comparison Banshie tries to offer to its clients.

7.3 Summary

Writing adapters for extractors to allow execution and evaluation in the Banshie framework requires some manual work, in case of the example integrations about 100 lines of codes each. Adapting an API to the extractor interface specification also differs greatly from extractor to extractor and highly depends on how easy it is to tokenize the input document, run the recognition process, iterate over annotated tokens and produce the desired XML output. Especially calculating the UTF-8 character offsets can be errorprone, as listing 23 illustrated.

Even though the example document and reference output were extremely small, the result is a nice example of a evaluation result one would expect to see a lot: Two different extractors with highly different quality-performance tradeoffs. OpenNLP only used 35 megabytes of memory and finished after about 2 seconds but only scored an F-measure of .5. CoreNLP on the other hand performed a perfect extraction but required more than 16 times the memory and 18 times the CPU time compared to OpenNLP.

Comparing and evaluating those results now highly depends on the use case, a fast and mediocre reliable extractor might be the perfect tool for one task, while another might require the best extraction, no matter the resource cost.

8 Conclusion

This chapter attempts to summarize this thesis by reviewing. It also gives an outlook for future work and research that can be done based on the results of this thesis.

8.1 Review

FiXme Fatal: Review

The main objective of this thesis, make an extensible, modular benchmarking framework, has been achieved. So, as a result, we have an Open Source Java-based framework which runs in any OSGi-compliant environment or embedded as a library in standalone applications.

8.2 Lessons learned

FiXme Fatal: Lessons learned

8.3 Outlook and future work

Banshie's architectural design is based on solid, state-of-the-art patterns, but to expand the frameworks capabilities beyond prototype character some ideas come to mind. Some of these ideas will be explained and discussed on the following pages.

The current focus is clearly the field of *Named Entity Recocgnition*, which is an important part of *Information Extraction* and *Natural Language Processing*, but there are other tasks in IE which are equally interesting, e.g. relationship extraction, part-of-speech tagging or grammatical sentence analysis (see chapter 2).

Supporting multiple IE tasks requires Banshie to offer a more flexible XML schema. Reusing the reference- and hypothesis schema definitions proposed by GeMap comes to mind [44].

Since Evaliex uses a different reference-hypothesis association algorithm, offering a swappable reference-hypothesis mapping algorithm for the performance evaluation could be a useful extension to the framework to compare results and/or to allow users to choose based on their use case.

The modified version of the *General Greedy Mapping Algorithm* used in *Evaliex* is based in string/word similarity. Future versions of the Banshie platform should support multiple algorithm, e.g. Levenshtein distance and Jaccard coefficient, as well as a plug-in mechanism for those similarity checks.

The current version of the framework only supports Java Virtual Machine (JVM)-based extractors and since different extractors have different requirements, e.g. memory and garbage collector configuration, applying custom additional command line parameters to the external Java process would be handy.

Another stage of expansion would include alternate `Engine` implementations to support non JVM-based extractors. Different engines would of course require different means to collect runtime events. In other words for different engines one needs to supply a viable JMX client alternative.

The framework, in its current state, is solely an API-based tool, which offers great embeddability for OSGi- and likewise Java SE environments. To support more use cases providing a simple text-based CLI seems to be a promising extension to the platform.

A Web User Interface (UI), in addition to the CLI, would be an even more user-friendly approach. A web-based frontend could include fail-safe, responsive and intuitive interface elements to allow easy upload, querying and execution of extractors. A Web UI would also be an excellent place to provide visual representations of statistical data and analytical results in the form of charts and diagrams.

Collecting, persisting and aggregating CPU time and memory consumption is the straigtforward approach to measure the runtime performance of a program. But other users might require different or additional measures like file system consumption, thread count or startup latency. JMX supports many, many more indicators and extensions to the *event production* implementation could support a more customizable approach in the future.

The memory consumption is calculated as the average heap size while the CPU time just looks at the last value. Those are just concrete implementations of generic aggregate functions: `AVG` and `LAST` in that case. A more sophisticated approach would be to support an extensible core set of aggregate functions, e.g. `MIN`, `MAX`, `AVG`, `STDDEV`, `VAR`, `SUM`, `FIRST` and `LAST`, which operate on the raw logging data. Even offering a lightweight MapReduce integration for a more flexible and user-oriented statistical analysis is imaginable.

References

[1] Joseph Mariani (Editor) Ronald Cole (Editor) Antonio Zampolli (Editor) Annie Zaenen (Editor). *Survey of the State of the Art in Human Language Technology.* Cambridge University Press, Mar. 1998.

[2] *Apache OpenNLP.* 2010. URL: http://opennlp.apache.org/.

[3] Douglas E. Appelt. „Introduction to information extraction". In: *AI Commun.* 12.3 (Aug. 1999), pp. 161–172. ISSN: 0921-7126. URL: http://dl.acm.org/citation.cfm?id=1216155.1216161.

[4] Alan Bateman. *JEP 102: Process API Updates.* 2011. URL: http://openjdk.java.net/jeps/102.

[5] *Bnd.* 2006. URL: http://www.aqute.biz/Bnd/Bnd.

[6] Kai-Uwe Carstensen et al. *Computerlinguistik und Sprachtechnologie: Eine Einführung.* 3. Aufl. Spektrum Akademischer Verlag, 2010. ISBN: 9783827414076.

[7] Kyle W. Cartmell. *Five Common java.lang.Process Pitfalls.* 2009. URL: http://kylecartmell.com/?p=9.

[8] Chia-Hui Chang et al. „A Survey of Web Information Extraction Systems". In: *IEEE Trans. on Knowl. and Data Eng.* 18.10 (Oct. 2006), pp. 1411–1428. ISSN: 1041-4347. URL: http://dx.doi.org/10.1109/TKDE.2006.152.

[9] Nancy Chinchor. „Four scorers and seven years ago: the scoring method for MUC-6". In: *Proceedings of the 6th conference on Message understanding.* MUC6 '95. Columbia, Maryland: Association for Computational Linguistics, 1995, pp. 33–38. ISBN: 1-55860-402-2.

[10] Nancy Chinchor. „MUC-4 evaluation metrics". In: *Proceedings of the 4th conference on Message understanding.* MUC4 '92. McLean, Virginia: Association for Computational Linguistics, 1992, pp. 22–29. ISBN: 1-55860-273-9.

[11] Nancy Chinchor. *MUC-7 Information Extraction Task Definition.* 1998. URL: http://www.itl.nist.gov/iaui/894.02/related_projects/muc/proceedings/ie_task.html.

[12] Nancy Chinchor. *What is Information Extraction?* 2001. URL: `http://www.itl.nist.gov/iaui/894.02/related_projects/muc/info/whats_ie.html`.

[13] Philipp Cimiano et al. „IE Evaluation Strategy“. Dot.Kom Deliverable D3-3. 2003. URL: `http://nlp.shef.ac.uk/dot.kom/pdocs/D3-3.pdf`.

[14] Michael Crystal et al. „A methodology for extrinsically evaluating information extraction performance“. In: *In Proc. of HLT/EMNLP*. 2005, pp. 652–659.

[15] Hamish Cunningham. „Information Extraction, Automatic“. In: *Encyclopedia of Language and Linguistics* 2 (2005).

[16] Hamish Cunningham et al. „GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications“. In: *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL'02)*. 2002.

[17] Hamish Cunningham et al. *Text Processing with GATE (Version 6)*. 2011. ISBN: 978-0956599315. URL: `http://tinyurl.com/gatebook`.

[18] George Demetriou et al. „ANNALIST – ANNotation ALIgnment and Scoring Tool“. In: *In: Proceedings of the Sixth International Conference on Language Resources and Evaluation, LREC 2008*. 2008.

[19] A. Douthat. „The Message Understanding Conference Scoring Software User's Manual“. In: 1998.

[20] Line Eikvil. *Information Extraction from World Wide Web - A Survey*. 1999.

[21] *Ellogon*. URL: `http://www.ellogon.org/`.

[22] Andrea Esuli and Fabrizio Sebastiani. „Evaluating information extraction“. In: *Proceedings of the 2010 international conference on Multilingual and multimodal information access evaluation: cross-language evaluation forum*. CLEF'10. Padua, Italy: Springer-Verlag, 2010, pp. 100–111. ISBN: 3-642-15997-4, 978-3-642-15997-8. URL: `http://dl.acm.org/citation.cfm?id=1889174.1889192`.

[23] Christina Feilmayr, Birgit Pröll, and Elisabeth Linsmayr. „EVALIEX - A Proposal for an Extended Evaluation Methodology for Information Extraction Systems“. In: *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)ce on Language Resources and Evaluation (LREC'12)*. Istanbul, Turkey: European Language Resources Association (ELRA), May 2012. ISBN: 978-2-9517408-7-7.

[24] Martin Fowler. *Inversion of Control Containers and the Dependency Injection pattern*. 2004. URL: http://www.martinfowler.com/articles/injection.html.

[25] Dayne Freitag and Nicholas Kushmerick. „Boosted Wrapper Induction“. In: AAAI Press, 2000, pp. 577–583.

[26] Robert Gaizauskas and Yorick Wilks. *Information Extraction: Beyond Document Retrieval*. 1998.

[27] *GATE Information Extraction*. URL: http://gate.ac.uk/ie/.

[28] Andrew Goldberg. „Advanced NLP: Automatic Summarization“. 2007.

[29] *Google Guice*. 2012. URL: https://code.google.com/p/google-guice/.

[30] R. Grishman. „Discovery methods for information extraction“. In: *ISCA & IEEE Workshop on Spontaneous Speech Processing and Recognition*. 2003.

[31] R. Grishman. „NLP: An information extraction perspective“. In: *AMSTERDAM STUDIES IN THE THEORY AND HISTORY OF LINGUISTIC SCIENCE SERIES 4* 292 (2007), p. 17.

[32] Ralph Grishman. „Information extraction: Techniques and challenges“. In: *Information Extraction A Multidisciplinary Approach to an Emerging Information Technology*. Ed. by MariaTeresa Pazienza. Vol. 1299. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1997, pp. 10–27. ISBN: 978-3-540-63438-6. DOI: 10.1007/3-540-63438-X_2. URL: http://dx.doi.org/10.1007/3-540-63438-X_2.

[33] Ralph Grishman and Beth Sundheim. „Design of the MUC-6 evaluation". In: *Proceedings of a workshop on held at Vienna, Virginia: May 6-8, 1996*. TIPSTER '96. Vienna, Virginia: Association for Computational Linguistics, 1996, pp. 413–422. DOI: `10.3115/1119018.1119072`. URL: `http://dx.doi.org/10.3115/1119018.1119072`.

[34] *Interface OperatingSystemMXBean*. 2013. URL: `http://docs.oracle.com/javase/6/docs/jre/api/management/extension/com/sun/management/OperatingSystemMXBean.html#getProcessCpuTime()`.

[35] Panos Ipeirotis. *Evaluating Information Extraction using xROC Curves*. 2007. URL: `http://www.behind-the-enemy-lines.com/2007/07/evaluating-information-extraction-using.html`.

[36] Mark Przybocki Jonathan et al. „Hub-4 Information Extraction Evaluation". In: *In Proceedings of the DARPA Broadcast News Workshop*. Morgan Kaufmann, 1999, pp. 13–18.

[37] *JSR 330: Dependency Injection for Java*. 2009. URL: `http://jcp.org/en/jsr/detail?id=330`.

[38] Kirk Knoernschild. *Java Application Architecture: Modularity Patterns with Examples Using OSGi (Robert C. Martin Series)*. 1st ed. Prentice Hall, Mar. 2012. ISBN: 9780321247131.

[39] In Knorz et al. *Automatic Document Classification: A thorough Evaluation of various Methods*. 2000.

[40] Andrew Lampert. *A Quick Introduction to Question Answering*. 2004.

[41] Alberto Lavelli et al. „Evaluation of machine learning-based information extraction algorithms: criticisms and recommendations". In: *Language Resources and Evaluation* 42.4 (2008), pp. 361–393.

[42] W. Lehnert et al. *Evaluating an Information Extraction System*. 1994.

[43] Linsmayr. „Evaliex - Information Extraction Evaluation Framework". Master thesis. Nov. 2010.

[44] Elisabeth Linsmayr. „GeMap Generic Mapper for XML Instances". Softwaredokumentation Version 2. 2010.

[45] Will Lowe and Gary King. „Some Statistical Methods for Evaluating Information Extraction Systems“. In: *Proceedings of the 10th Conference of the European Chapter of the Association for Computational Linguistics* (2003), pp. 19–26.

[46] John Makhoul et al. „Performance Measures For Information Extraction“. In: *In Proceedings of DARPA Broadcast News Workshop*. 1999, pp. 249–252.

[47] Diana Maynard et al. „Ontology-based information extraction for market monitoring and technology watch“. In: *In ESWC Workshop "End User Apects of the Semantic Web*. 2005.

[48] Günter Neumann. „Informationsextraktion“. In: *Computerlinguistik und Sprachtechnologie - Eine Einführung* (2001).

[49] Nist. *Automatic Content Extraction 2008 Evaluation Plan*. 2008. URL: `http://www.itl.nist.gov/iad/mig/tests/ace/2008/doc/ace08-evalplan.v1.2d.pdf`.

[50] *OSGi Alliance Specifications*. URL: `http://www.osgi.org/Specifications/HomePage` (visited on 07/10/2012).

[51] *OSGi Service Platform Release 4 Version 4.2 Enterprise Specification*. URL: `http://www.osgi.org/Download/Release4V42`.

[52] Thierry Poibeau and Cédric Messiant. *Do we still Need Gold Standards for Evaluation?* 2008.

[53] Gunnar Rätsch. *A Brief Introduction into Machine Learning*. 2004.

[54] C.J. van Rijsbergen and Ph. D. *Information Retrieval*. 1979.

[55] Sunita Sarawagi. „Information Extraction“. In: *Found. Trends databases* 1.3 (Mar. 2008), pp. 261–377. ISSN: 1931-7883. DOI: `10.1561/1900000003`. URL: `http://dx.doi.org/10.1561/1900000003`.

[56] Yutaka Sasaki. *The truth of the F-measure*. 2007.

[57] Marcus Schramm. „Informationsextraktion auf Basis strukturierter Daten“. MA thesis. TU Dresden, 2008.

[58] Christian Siefkes. *An Incrementally Trainable Statistical Approach to Information Extraction: Based on Token Classification and Rich Context Model*. Vdm Verlag Dr. Müller, July 2008. ISBN: 9783639001464.

[59] Christian Siefkes and Peter Siniakov. „An Overview and Classification of Adaptive Approaches to Information Extraction“. In: *JOURNAL ON DATA SEMANTICS, IV:172–212. LNCS 3730*. Springer, 2005.

[60] An De Sitter, Tood Calders, and Walter Daelemans. „A Formal Framework for Evaluation of Information Extraction“. In: *A3* (Apr. 2004).

[61] Mark Stamp. *Once Upon a Time-Memory Tradeoff*. 2003.

[62] *Stanford CoreNLP*. 2012. URL: `http://nlp.stanford.edu/software/corenlp.shtml`.

[63] Jordi Turmo, Alicia Ageno, and Neus Català. „Adaptive information extraction“. In: *ACM Comput. Surv.* 38.2 (July 2006). ISSN: 0360-0300. DOI: `10.1145/1132956.1132957`. URL: `http://doi.acm.org/10.1145/1132956.1132957`.

[64] J. Weinhofer. „Extraktion semantisch relevanter Daten aus natürlich sprachlichen Inhalten in Hinblick auf eine automatische Fragengenerierung“. MA thesis. Technische Universität Graz, 2010.

[65] Wikipedia. *Information Extraction*. 2012. URL: `http://en.wikipedia.org/wiki/Information_extraction` (visited on 01/19/2013).

[66] Wikipedia. *Modularity*. 2012. URL: `http://en.wikipedia.org/wiki/Modularity#Modularity_in_technology_and_management`.

[67] Wikipedia. *Precision and recall*. 2012. URL: `http://en.wikipedia.org/wiki/Precision_and_recall`.

[68] Yorick Wilks. „Information extraction as a core language technology“. In: *Information Extraction A Multidisciplinary Approach to an Emerging Information Technology*. Ed. by MariaTeresa Pazienza. Vol. 1299. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1997, pp. 1–9. ISBN: 978-3-540-63438-6. DOI: `10.1007/3-540-63438-X_1`. URL: `http://dx.doi.org/10.1007/3-540-63438-X_1`.

[69] Osmar R. Zaïane. *Introduction to Data Mining*. 1999.

Acronyms

ACE Automatic Content Extraction. 13

ANNALIST Annotation Alignment and Scoring Tool. 34

API Application Programming Interfrace. 33, 35, 44, 71, 74, 76

ASF Apache Software Foundation. 31, 66

BDUF Big Design Up Front. 37

CLI Command Line Interface. 35, 76

CO Coreference Resolution. 8

DARPA Defense Advanced Research Projects Agency. 6

DI Dependency Injection. 37, 39–41

DIMA Database Systems and Information Management. 1

EDR Entity Detection and Recognition. 13

GATE General Architecture for Text Engineering. 33

HMM Hidden Markov Models. 11

IE Information Extraction. 1, 2, 4, 6, 8, 13, 15, 17–19, 24, 33, 75

IoC Inversion of Control. 37

JEP JDK Enhancement-Proposal. 53

JMX Java Management Extensions. 56, 57, 76

JPA Java Persistence API. 41, 42, 44, 58

JSON JavaScript Object Notation. 57, 58

JVM Java Virtual Machine. 75, 76

KDD Knowledge Discovery in Databases. 16

MUC Message Understanding Conferences. 6, 13

NER Named Entity Recognition. 8, 66, 68, 69, 72

NLP Natural Language Processing. 4, 15, 66

OBIE Ontology-Based Information Extraction. 13

OCR Optical Character Recognition. 13

OSGi Open Services Gateway initiative. 2, 29–32, 41, 42, 76, 86

PPV Positive Predictive Value. 20

RDBMS Relational database management system. 41

RDR Relation Detection and Recognition. 13

ST Scenario Template Production. 10

SVM Support Vector Machines. 11

TDD Test-Driven Development. 28

TE Template Element Construction. 9

TR Template Relation Construction. 9

UI User Interface. 76

VDR Event Detection and Recognition. 13

WG Wrapper Generation. 12

XML Extensible Markup Language. 48, 74

List of Figures

List of Tables

List of Listings