



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

Thesis Advisor: Prof. Dr. Stefan Edlich

Second Assessor: Prof. Dr. Agathe Merceron

March 19, 2013

*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

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. *Information Extraction (IE)* aims 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. Critically evaluating proposed information extraction algorithms is crucial to ensure a continuous improvement in system performances.

This thesis aims to contribute to the process of automation and standardization of the IE evaluation process. It does so by comparing different IE tasks and approaches for evaluating corresponding systems. The findings were then implemented into a modular framework which does not only support IE performance analysis but also provide benchmarking capabilities to measure runtime performances. The practicability of the framework will be shown by implemented adapters for different Open Source extraction systems and perform evaluations.

## **Zusammenfassung**

Im Zeitalter des Internets und Social-Media-Seiten werden täglich riesige Mengen von unstrukturierteren Daten produziert. Wesentlich mehr, als sich manuell noch weiterverarbeiten geschweige denn erfassen lassen. *Information Extraction* versucht Techniken und Algorithmen zu definieren, entwickeln und zu testen um Informationen aus unstrukturierten oder semistrukturierten Datenquellen zu extrahieren und in eine Darstellung zu überführen die für eine spätere Analyse und Weiterverarbeitung besser geeignet ist. Die kritische Bewertung von IE-Algorithmen ist entscheidend, um eine kontinuierliche Verbesserung der Extraktionsqualität zu gewährleisten.

Diese Arbeit zielt darauf ab, zur Automatisierung und Standardisierung des IE-Evaluationsprozesses beitragen. Dazu werden verschiedene IE-Aufgaben und Ansätze für die Beurteilung entsprechender Systeme aufgezeigt und beurteilt. Die Ergebnisse werden dann in Form eines modularen Framework umgesetzt, das nicht nur die Extraktionsleistung von IE-Systemen, sondern auch das Laufzeitverhalten bewertet. Die Praxistauglichkeit des Frameworks wird anhand der Integration und Evaluation von Open Source Extraktoren gezeigt.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Background . . . . .	1
1.2	Challenges . . . . .	1
1.3	Objectives . . . . .	2
1.4	Motivation . . . . .	2
1.5	Structure . . . . .	3
<b>2</b>	<b>Information Extraction</b>	<b>4</b>
2.1	Definition . . . . .	4
2.2	History of Information Extraction . . . . .	5
2.2.1	Message Understanding Conferences . . . . .	6
2.2.2	Automatic Content Extraction . . . . .	7
2.3	Most typical tasks . . . . .	7
2.3.1	Named Entity Recognition . . . . .	8
2.3.2	Coreference Resolution . . . . .	8
2.3.3	Template Element Construction . . . . .	9
2.3.4	Template Relation Construction . . . . .	9
2.3.5	Scenario Template Production . . . . .	10
2.4	Development and methods of IE . . . . .	10
2.4.1	Knowledge Engineering . . . . .	10
2.4.2	Machine Learning . . . . .	11
	Supervised learning . . . . .	11
	Semi-supervised learning . . . . .	11
	Unsupervised learning . . . . .	11
2.4.3	Web information extraction . . . . .	11
2.5	Related fields . . . . .	12
2.5.1	Information Retrieval . . . . .	12
2.5.2	Automatic summarization . . . . .	13
2.5.3	Document classification . . . . .	13
2.5.4	Data Mining . . . . .	14
2.5.5	Question answering . . . . .	14
2.6	Summary . . . . .	14
<b>3</b>	<b>Formal evaluation methodology</b>	<b>16</b>

3.1	IE approaches . . . . .	16
3.1.1	One Best per Document . . . . .	16
3.1.2	All Occurences . . . . .	17
3.2	Performance measures . . . . .	19
3.2.1	Matching rules . . . . .	20
3.2.2	Precision . . . . .	20
3.2.3	Recall . . . . .	21
3.2.4	F-measure . . . . .	22
3.2.5	Error measure . . . . .	22
3.2.6	Error per Response Fill . . . . .	23
3.2.7	Slot Error Rate . . . . .	23
3.3	Discussion . . . . .	23
3.4	Runtime performance measures . . . . .	25
3.4.1	CPU time . . . . .	25
3.4.2	Memory consumption . . . . .	25
3.5	Summary . . . . .	25
<b>4</b>	<b>Modularity</b>	<b>27</b>
4.1	Module definition . . . . .	27
4.2	OSGi . . . . .	28
4.2.1	Implementations . . . . .	29
4.3	Relevant principles and design patterns . . . . .	30
4.3.1	Single Responsibility Principle . . . . .	30
4.3.2	Dependency Inversion Principle . . . . .	31
4.3.3	Dependency Injection . . . . .	31
4.4	Supporting libraries and tools . . . . .	32
4.4.1	Guice . . . . .	32
4.4.2	Guice Extensions . . . . .	33
	Peaberry . . . . .	33
4.4.3	BND Tool and the Maven Bundle Plugin . . . . .	34
4.5	Summary . . . . .	35
<b>5</b>	<b>Related Work</b>	<b>36</b>
5.1	Evaliex . . . . .	36
5.2	GATE . . . . .	36
5.3	Ellogon . . . . .	37

5.4	ANNALIST . . . . .	38
5.5	Summary . . . . .	38
<b>6</b>	<b>Design</b>	<b>40</b>
6.1	Analysis and requirements . . . . .	40
6.2	Architecture . . . . .	40
6.3	API . . . . .	43
	Domain model and persistence . . . . .	43
	Execution . . . . .	44
	Evaluation . . . . .	44
	API Usage . . . . .	45
6.4	Extractor interface specification . . . . .	46
6.5	Reference-hypothesis association . . . . .	48
6.6	Implementation . . . . .	49
6.6.1	Engine . . . . .	49
	Process management . . . . .	50
	Process monitoring . . . . .	52
	Event production . . . . .	55
	Event persistence . . . . .	56
6.6.2	Quality evaluation . . . . .	58
6.6.3	Performance evaluation . . . . .	60
6.7	Implementation obstacles . . . . .	62
6.7.1	Persistence . . . . .	63
6.8	Summary . . . . .	63
<b>7</b>	<b>Results</b>	<b>65</b>
7.1	Open Source Extractors . . . . .	65
7.1.1	Apache OpenNLP . . . . .	65
7.1.2	Stanford CoreNLP . . . . .	66
7.2	Examples . . . . .	66
7.2.1	Apache OpenNLP . . . . .	67
7.2.2	Stanford CoreNLP . . . . .	70
7.3	Comparison . . . . .	72
7.4	Summary . . . . .	73
<b>8</b>	<b>Conclusion</b>	<b>74</b>



8.1	Review . . . . .	74
8.2	Lessons learned . . . . .	75
8.3	Outlook and future work . . . . .	76
<b>References</b>		<b>78</b>
<b>List of Figures</b>		<b>85</b>
<b>List of Tables</b>		<b>86</b>
<b>List of Listings</b>		<b>87</b>

# 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 Information Extraction (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 [63]. 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. To 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.

As an example, let's assume someone implemented an extraction tool to perform an automatic product announcement feed aggregation. Such a system could work in conjunction with a web crawler which scans websites, like company blogs or homepages, and passes the content on to the extractor. The extractor then attempts to annotate words in the text, that indicate a product announcement, like the name, description, date and estimated price. If different, competing parties would propose such a system, our software could be used to compare those. Our framework performs an extraction for each extractor by passing them a test document and then calculates the results by comparing the output to the reference. The client is then able to compare extraction quality and runtime performance of different extractors.

### 1.4 Motivation

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

---

<sup>1</sup><http://www.dima.tu-berlin.de/>

<sup>2</sup><http://www.mia-marktplatz.de/>

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 relying on the Java programming language and the Java platform in general.

## 1.5 Structure

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

Chapter 2 (Information Extraction) contains different definitions of IE, a 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 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 an 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 [67][45].

### 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 [13]

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 [58]

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 [68]

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. [44][51][61].

The challenge in information extraction according to [32] 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 [2][32][33][45].

## **2.2 History of Information Extraction**

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 [61][21]. The first application of information extraction occurred in the 1950s, where the information from texts were reduced into a table structure. [34][28][71].

### 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 [35][34].

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 [66][4][16][45]:

	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 [14][44].

### 2.2.2 Automatic Content Extraction

The program of the Automatic Content Extraction (ACE) <sup>1</sup> 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 [52][43][66][45]:

- **Entity Detection and Recognition (EDR)**  
Identification of entity types and subcategories
- **Relation Detection and Recognition (RDR)**  
Recognition of relationships between entities
- **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 [16].

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 [7][43]:

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. [16]

---

<sup>1</sup><http://www.itl.nist.gov/iad/mig/tests/ace/>



### 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 [45].

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 [58].

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 [58].

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 the 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 [58].

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 [58].

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 [16].

Regarding the given example document, ST discovers that there was a rocket launching event on tuesday in which the various entities were involved [16].

## **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 [60].

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 [45].

### **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 [60][66].

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 [4].

### **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 [7][60]:

#### **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 [60][62][7].

#### **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 [7][9].

#### **Unsupervised learning**

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 [7][60]. A possible use case for unsupervised machine learning in the context of IE is shown by [1], which proposed a method of unsupervised relation extraction.

### **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 [21][27][45].

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 [9][21]. The wrapper generation focuses on the integration of multiple sources and counteracts the heterogeneity of the web by using a wrapper library [61][66].

## **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 [57].

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. [29]

### **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 [30].

### **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 [41].

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 [41].

### 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 [72].

### 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 [42].

## 2.6 Summary

This chapter tried to give an overview of the field of IE by defining the term *information extraction* and giving a discourse about its history and programmes, a more or less complete list of the most typical tasks and 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.

f



## 3 Formal evaluation methodology

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 [45]. 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 definitions for the most frequently used IE approaches, „All Occurrences“ and „One Best per Document“ and 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. 40ff.).

### 3.1 IE approaches

Depending on the intended use case of the IE system, different approaches to information extraction may be useful.

#### 3.1.1 One Best per Document

A first possibility is the One Best per Document (OBD) approach. In this case, we see the *IE* task as filling a given template structure. This is for example useful if the goal of the extractor is to fill a database in order to be able to query the information contained in the documents. [63]. Templates are organized in slots which are filled during extraction time. Table 5 shows an example template for the job advertisements domain.

Slot	Value
<b>title</b>	Chief Business Officer
<b>salary</b>	80,000 USD
<b>company</b>	Acme Corporation
<b>due-date</b>	February 5, 2013

**Table 5:** Example template for the OBD approach

For evaluating IE systems using the OBD approach, the exact location of the annotations in the document is not important. [63]

### 3.1.2 All Occurences

A second possibility is the All Occurences (AO) apporach. In this case, every occurence of certain items in a document need to be found. This is usually realized by annotating or tagging tokens in the original text. [63]

## JOB OPPORTUNITY

The <company>Acme Corporation</company> requires the service of qualified young persons to fill up the following positions

- <title>Chief Business Officer</title>
- <title>Finance Manager</title>

### Qualification

MBA from a recognize educational institution.

### Experience

5 years post qualification experience in a similar position or in the rank of Assistant in an organization of repute.

### Age

Between 24 and 35 years with excellent health.

### Salary

The <company>Acme Corporation</company> offers 80,000 per annum.

Interested candidates may send their hand written applications along with a copy of CV, two passport size potographs and names and contact details of three referees by or before <due-date>18 May, 2013</due-date>.

**Figure 1:** Annotated example document for the AO approach

The AO IE task for the item ,company‘ for the document in figure 1 gives the results *Acme Corporation* and *Acme Corporation*, while the task for ,title‘ would return *Chief Business Officer* and *Finance Manager*. Note that in the first case (,company‘) both answers indicate different occurences of the same company. In the second case (,title‘), both answers indicate a different job title. But since we want to find every occurence of every instance, this doesn’t matter in the formal setting [63].

The framework, developed as part of this thesis, focuses on this approach when evaluating information extraction systems.

### 3.2 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 6 shows a confusion matrix [63].

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

**Table 6:** 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 [63] 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 [37].

Another way to calculate performance measures is based on the notation proposed by Makhoul et al. [48]:

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)

### 3.2.1 Matching rules

The way of deciding if an extracted entity („prediction“) is correct or not, is crucial in computation of the scores [63]. D. Freitag gives three different possibilities [26]:

- *exact rule*: a prediction is correct, if it's exactly equal to an answer. E.g. if the answer given is ‚Albert Einstein‘, ‚Dr. Albert Einstein‘ would not be correct.
- *contain rule*: a prediction is correct, if it contains an answer and maybe some additional neighboring tokens. E.g. if the answer given is ‚Albert Einstein‘, ‚Dr. Albert Einstein‘ would be correct as well.
- *overlap rule*: a predication is correct, if it contains a part of an answer and maybe some additional neighboring tokens. E.g. if the answer given is ‚Dr. Albert Einstein‘, ‚Albert Einstein‘ would be correct as well.

Which rule is preferable, depends heavily on the use case of the IE system. If an IE system is filling a database which answers queries, the exact rule might return better results. On the other hand, if someone's task is to find all job titles in a set of documents, a system that gives a prediction overlapping with the correct answer will be more useful than one that only returns the exact answer for a smaller share of the documents. [63]

The framework, developed as part of this thesis, will be using mixed approach of the *contain* and *overlap* rule.

### 3.2.2 Precision

The *precision* ( $\pi$  or P), also called Sensitivity, is defined as the percentage of correctly retrieved data in the hypothesis [7].

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

**Figure 2:** Precision formula

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

### 3.2.3 Recall

The *recall* ( $\rho$  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 [7].

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

**Figure 3:** Recall formula

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

Figure 4 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) [70].



**Figure 4:** Recall and precision example figure [70]

### 3.2.4 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 5:** F-measure formula

Figure 6 shows the more general formula of  $F_\beta$ -score, which contains a parameter  $\beta$  to control the balance between *precision* and *recall*. When  $\beta = 1$ ,  $F_1$  comes to be equivalent to the harmonic mean of  $\pi$  and  $\rho$ . If  $\beta > 1$ ,  $F$  becomes more recall-oriented and if  $\beta < 1$ , it becomes more precision-oriented [59].

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

**Figure 6:**  $F_\beta$ -score formula [11]

$F_\beta$  measures the effectiveness of retrieval with respect to a user who attaches  $\beta$  times as much importance to recall as precision [57].

Another commonly used formula for the F-measure is shown in figure 7. In contrast to Rijsbergen's  $F_\beta$ -score formula, the balance parameter  $\alpha$  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 7:**  $F_\alpha$ -score formula [48]

### 3.2.5 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* [48]:

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

**Figure 8:** Error measure formula

### 3.2.6 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  $\alpha$  weights [10][48].

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

**Figure 9:** Error per response fill formula

### 3.2.7 Slot Error Rate

The *Slot Error Rate* (*SER*) was originally proposed by Makhoul et al. [48] 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 10:** Slot Error Rate formula

## 3.3 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 [63].

For  $\alpha = 0.5$ , E in 8 reduces to:

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

**Figure 11:** Error measure for  $\alpha = 0.5$

The denominator in figure 11 is equal to the average of the number of slots in the reference and in the hypothesis. But the major effect in 11 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. [48] 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 [10][48] as shown in figure 9.

ERR removes the deweighting of D and I by simply removing the  $\alpha$  weights in 8. 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 9 as [48]:

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

**Figure 12:** 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 [48].

## 3.4 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.

### 3.4.1 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.

### 3.4.2 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 occasionally the peak value might be of a certain interest as well.

## 3.5 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. 40ff.).

## 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 [40]

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 [69]

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 [40]

Figure 13 illustrates this definition and all the individual aspects of a module [40].

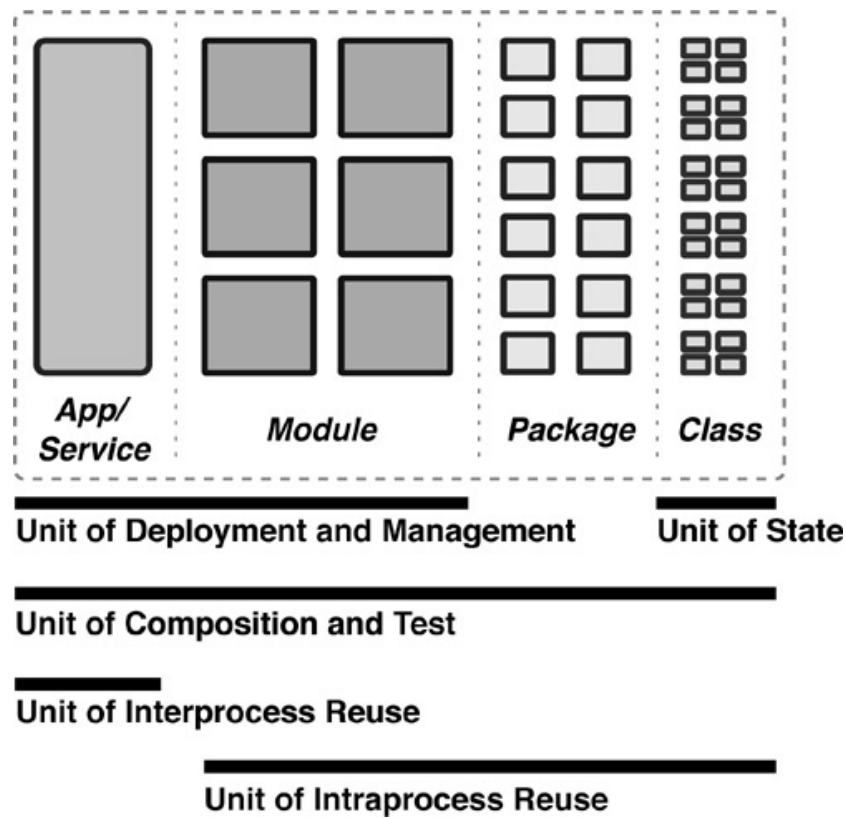


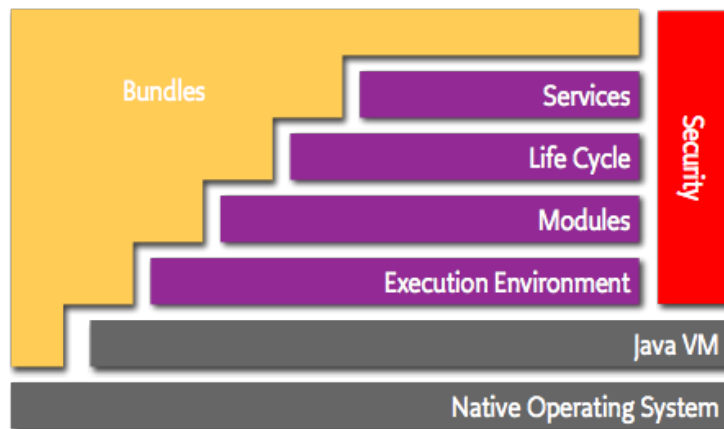
Figure 13: Module definition diagram

## 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 [40].

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 [53]. 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 [40]. Figure 14 shows the layered model architecture of the OSGi service platform.



**Figure 14:** OSGi layered model [53]

#### 4.2.1 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.

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.

## **4.3 Relevant principles and design patterns**

Some of the most important aspects of modular software design, in the form of principles and design patterns, will be discussed on the following pages. The findings derived from these patterns influenced the way the framework was designed and implemented, as shown in chapter 6 (pp. 40ff).

### **4.3.1 Single Responsibility Principle**

The term *Single Responsibility Principle* was first introduced by Robert C. Martin in his book *Agile Software Development, Principles, Patterns, and*

*Practices* [49]. It states that every class should have one, and only one, responsibility. The principle is originally coined for Object Oriented Design (OOD) but can easily be adapted to module design. The reason for keeping a class or module focused on a single concern is that it makes the class or module more robust, since changing a class increases the danger of other parts of it to break. Applied to concrete module design one could say that if a module is concerned with two aspects of a problem it should be separated into two different modules.

#### 4.3.2 Dependency Inversion Principle

The *Dependency Inversion Principle* again was postulated by Robert C. Martin and states that High-level modules should not depend on low-level modules. Both should depend on abstractions [49]. The problem with the traditional approach, higher-level components depending on lower-level components, limits the higher-level components reusability because the modules are too tightly coupled. The dependency inversion principle allows to increase the reusability of higher-level modules by allowing to choose from different lower-level implementations. In the context of module design it's important to depend on modules with a well-defined interface. The best way to enforce the replaceability of lower-level modules is to introduce interface modules which solely contain the Application Programming Interface (API) for a component. Implementors of this interface can then be swapped during runtime. In concrete systems this usually results in API and implementation modules. Modules should then only depend on API modules but never on implementation modules.

#### 4.3.3 Dependency Injection

Dependency Injection (DI) is way too solve the problem described by the *Dependency Inversion Principle*. DI is an expression introduced by Martin Fowler in its article *Inversion of Control Containers and the Dependency Injection Pattern* [25]. 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 [39]. 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 [39].

The main reason why DI works well with modularity is the fact, that its main purpose is dependency management which is also very important when designing and implementing modular systems. DI forces clients to be honest about their dependencies instead of hiding them in implementation details, which allows clients using modules to quickly get a realistic view of what the module is supposed to do and what dependencies it requires to do so.

## 4.4 Supporting libraries and tools

There are several freely available Java libraries and build tools that solve some of the problems when designing modular software in general and for the OSGi platform in particular. All of the libraries and tools described in this chapter will be used in the design and implementation of the framework.

### 4.4.1 Guice

Guice is a lightweight dependency injection framework for Java [31]. 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.

```
final class DefaultEngine implements Engine {  
  
    private final ProcessService service;  
  
    @Inject  
    DefaultEngine(ProcessService service) {  
        this.service = service;  
    }  
  
}
```

**Listing 2:** Constructor injection

#### 4.4.2 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<sup>1</sup> and Multibindings<sup>2</sup>. 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<sup>3</sup>, 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

---

<sup>1</sup><https://code.google.com/p/google-guice/wiki/AssistedInject>

<sup>2</sup><https://code.google.com/p/google-guice/wiki/Multibinding>

<sup>3</sup><https://code.google.com/p/peaberry/>

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.

```
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.

#### 4.4.3 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 metadata 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

[6]. 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 <sup>1</sup>. 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 5:** Maven Bundle Plugin usage

## 4.5 Summary

This chapter focused on the modularity aspect of this thesis. We discussed the concept of modularity, how it relates to module design and why OSGi is the best available module system for the Java platform at the moment. Additionally some relevant patterns, principles and libraries, which are closely related to modular software design and implementation, were described and discussed.

---

<sup>1</sup><http://felix.apache.org/site/apache-felix-maven-bundle-plugin-bnd.html>

## 5 Related Work

This chapter aims to provide an 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 [24].

*Evaliex* was part of a master thesis by Linsmayr in 2010: „Evaliex - Information Extraction Evaluation Framework“ [45]. 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“ [24].

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 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)<sup>1</sup> is a Free & Open-Source infrastructure for developing and deploying software components that processes human language. It is more than 15 years old and in active use for

---

<sup>1</sup><http://gate.ac.uk/>

all types of computational tasks involving language. GATE excels at text analysis of all shapes and sizes. [18].

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 [45].

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<sup>1</sup> 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 [22].

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 [45].

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.

---

<sup>1</sup><http://www.ellogon.org/>

## 5.4 ANNALIST

Annotation Alignment and Scoring Tool (ANNALIST)<sup>1</sup> 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 [19].

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 [45].

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

---

<sup>1</sup><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 evaluation 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 DI, Inversion of Control (IoC) and Composition over Inheritance in mind.

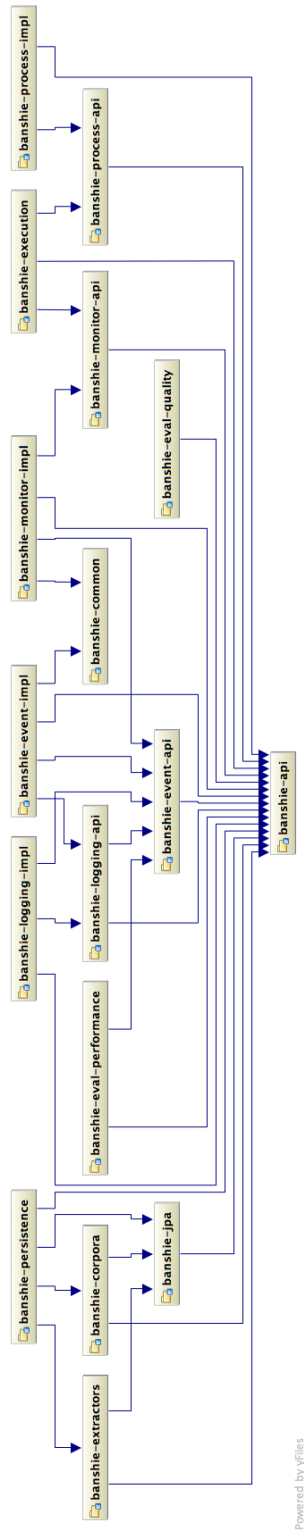
Since Banshie had to be developed in an Open Source fashion to support broader use cases, its whole code base as well as the documentation is available via

<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. Knörnschild 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)* [40]

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.



Powered by yfiles

**Figure 15:** Module dependencies

The most important module, as shown in figure 15 is the *banshie-api* module. It contains the public API of the framework which itself is described in chapter 6.3 (see pp. 43ff.). The API module is the only module clients or users of the framework should directly depend on. All the other modules form the implementation of the framework's public API and are highly implementation specific.

The next most important modules are *banshie-corpora*, *banshie-extractors*, *banshie-execution*, *banshie-eval-performance* and *banshie-eval-quality*. They are based on the package hierarchy of the API and provide the default implementation of their corresponding packages.

Since a single implementation module for some of these packages would violate the *Single Responsibility Pattern* (s. chapter 4.3) they had to be broken up into smaller ones. The results of this process are the following modules: *banshie-process-api*, *banshie-process-impl*, *banshie-monitor-api*, *banshie-monitor-impl*, *banshie-event-api*, *banshie-event-impl*, *banshie-logging-api* and *banshie-logging-impl*. The naming schema indicates that all of these modules have been split up into their own respective API and implementation modules, which can be exchanged by alternative versions if desired. The API modules form an internal API which allows implementation modules to communicate through a well-defined interface, those modules must not be confused with the public API module *banshie-api*. The different responsibilities and respective APIs will be discussed in chapter 6.3. It should also be noted, that figure 15 nicely demonstrates the result of applying the patterns discussed in chapter 4.3: Every module only depends on API (or library) modules, but never on implementation modules.

Another group of modules contains the *banshie-common*, *banshie-jpa* and *banshie-persistence* modules. The Java Persistence API module provides persistence-related utility features while the common module just offers more general framework-related features. The persistence module is not so much the result of a pattern-based design approach but rather the consequence of OSGi's bad support for Java Persistence API (JPA). Aggregating domain model and persistence-related classes into one module makes integrating them into an OSGi container much easier, since all relevant classes are loaded from a single classloader.

## 6.3 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.

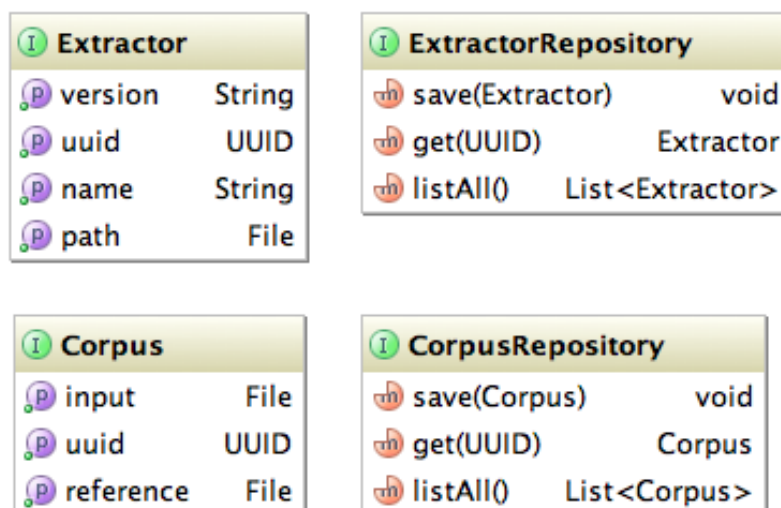


Figure 16: Banshie model and persistence API

## 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 6.6.

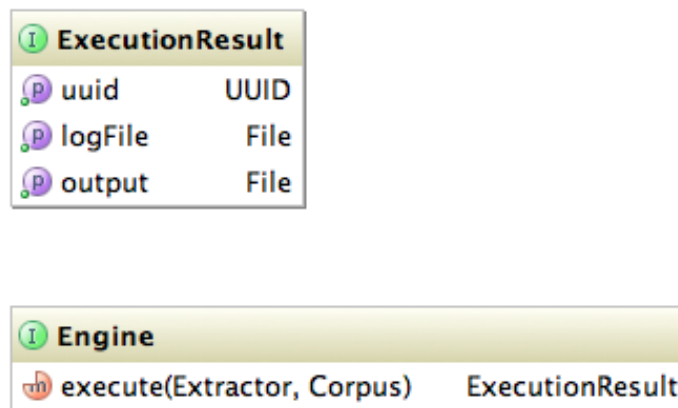













Figure 17: Banshie Execution API


## 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 18:


E Dimension	
 CPU_TIME	Dimension
 MEMORY_CONSUMPTION	Dimension
 TIME	Dimension
 PRECISION	Dimension
 RECALL	Dimension
 F_MEASURE	Dimension
 ERROR_MEASURE	Dimension
 ERROR_PER_RESPONSE_FILL	Dimension
 SLOT_ERROR_RATE	Dimension
 name	String
 toString()	String

I QualityEvaluator	
 evaluate(File, File)	Map<Dimension, Value>

I PerformanceEvaluator	
 evaluate(File)	Map<Dimension, Value>

I Value	
 value	double

**Figure 18:** 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 6 shows the simple basic steps required to perform a single extractor execution and evaluation.

```

// 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 6:** Banshie API usage

## 6.4 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*<sup>1</sup> 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:

---

<sup>1</sup><https://code.google.com/p/jarjar/>

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

**Listing 7:** 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 8.

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

```
<?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 8:** Banshie XML Schema



As shown in listing 9, 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 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  $E = mc^2$  (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 9: 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 [35].

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.

## 6.5 Reference-hypothesis association

Douthat proposed the original version of the „General Greedy Mapping Algorithm“ in 1998 [20]. It's based on finding matching pairs of spans in the reference and the predication. Evaliex even extended this algorithm by basing the matching on string or word similarity algorithms like Levenshtein-distance or the Jaccard-coefficient [45]. But since Banshie, in its current version, focuses

solely on the *All Occurences* approach of IE (see 3.1, pp. 16ff.), 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 (*overlap* and *contains* rule) of spans defined in the reference to score in the evaluation metrics. See 3.2.1, p. 20, for a detailed explanaiton of the different matching rules.

## 6.6 Implementation

### 6.6.1 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 [40] and discussed in chapter 4. The engine's submodules are described in the following chapters.

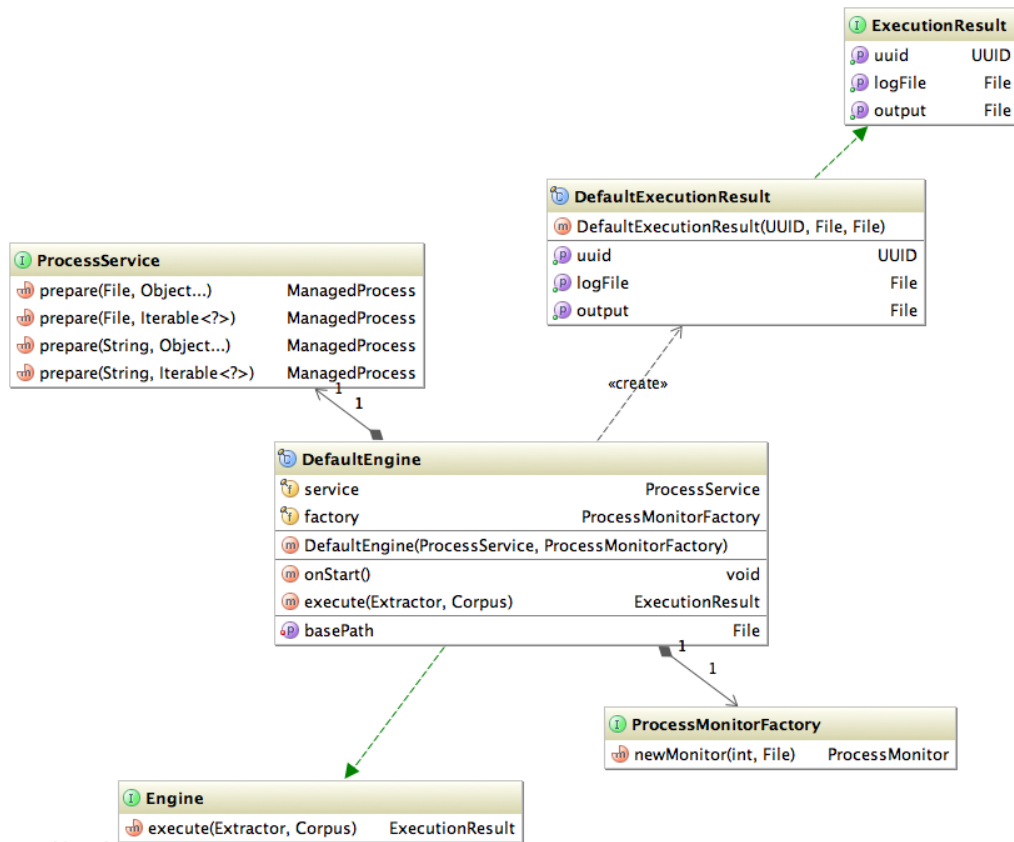


Figure 19: Engine implementation

## 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 [8]. There even has been filed a JDK Enhancement-Proposal (JEP) to improve the API for controlling and managing operating system processes. [5].

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 10 demonstrates the minimal steps to use the

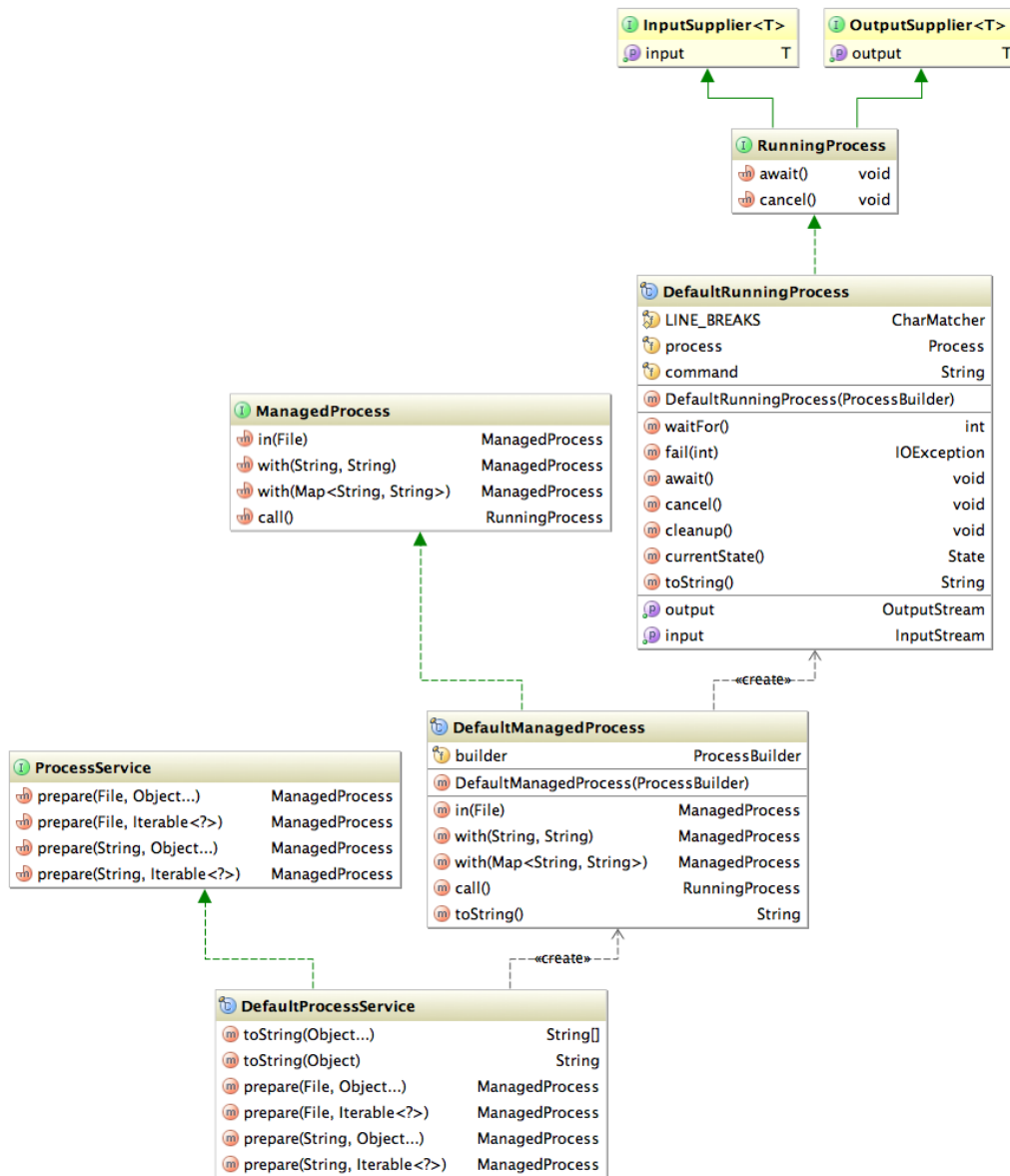
framework's Process API while figure 20 shows the relevant parts of it.

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

// write to process.getOutputStream() or
// read from process.getInputStream()

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

**Listing 10:** ProcessService API example usage

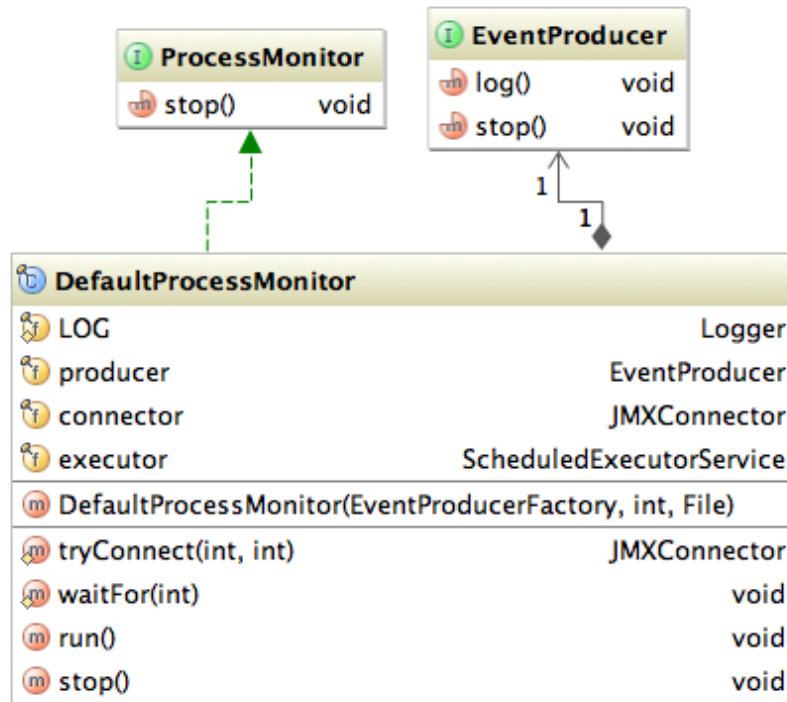


**Figure 20:** ProcessService implementation

## 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 21:** ProcessMonitor implementation

The repeated polling is realized with the JDK's built-in `ScheduledExecutorService`<sup>1</sup>, which schedules a special `Runnable` to run in a scheduled fashion, like shown in listing 11

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

**Listing 11:** Scheduling in java

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 12.

<sup>1</sup><http://docs.oracle.com/javase/6/docs/api/java/util/concurrent/ScheduledExecutorService.html>

```

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 12:** 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);

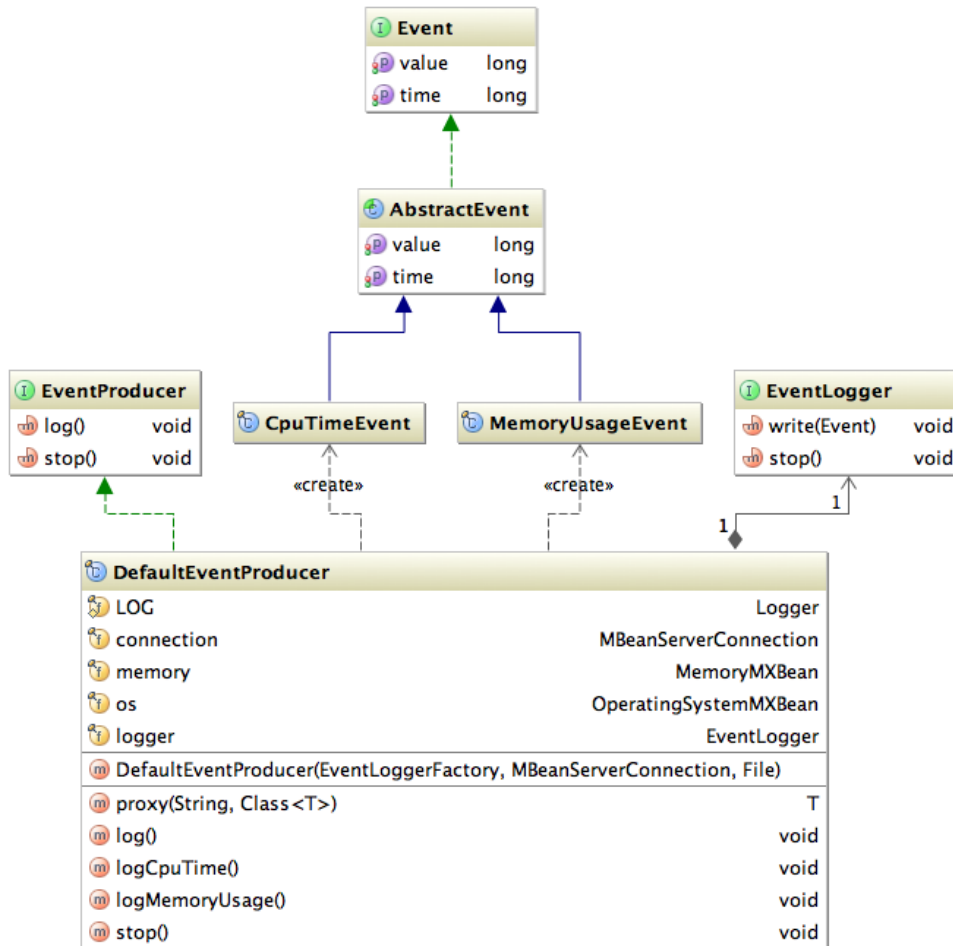
```

**Listing 13:** 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 22:** 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**<sup>1</sup> and **OperatingSystemMXBean**<sup>2</sup> classes, creates

<sup>1</sup><http://docs.oracle.com/javase/6/docs/api/java/lang/management/MemoryMXBean.html>

<sup>2</sup><http://docs.oracle.com/javase/6/docs/jre/api/management/extension/com/sun/management/OperatingSystemMXBean.html>



`CpuTimeEvents` and `MemoryUsageEvents` 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. `Events` are serialized using JSON mapping capabilities of the Jackson<sup>1</sup> 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.

---

<sup>1</sup><http://jackson.codehaus.org/>

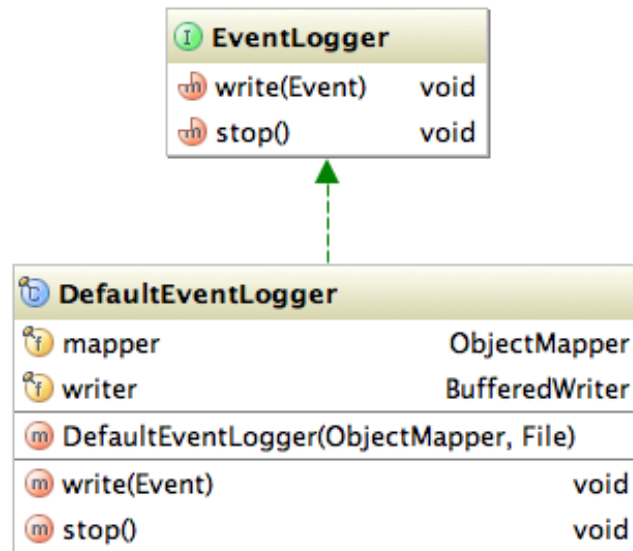


Figure 23: EventLogger implementation

```

{"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 14: Example event log file excerpt

## 6.6.2 Quality evaluation

The default `QualityEvaluator` implementation has several 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.

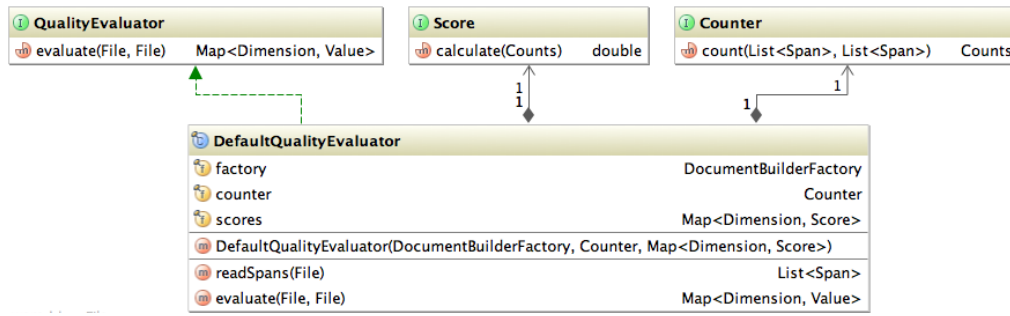


Figure 24: QualityEvaluator implementation

The reference-hypothesis association is provided by the `Counter` class, as shown in figure 25. For details about the reference-hypothesis association algorithm used, please consult chapter 6.5.

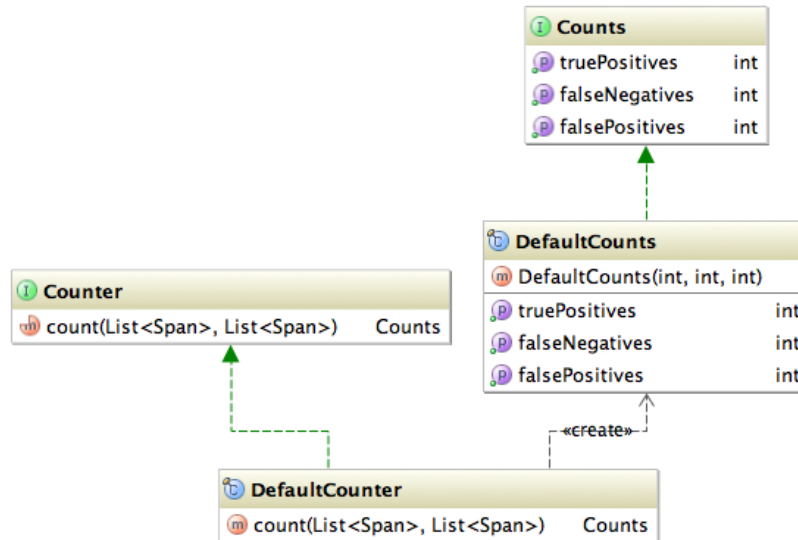


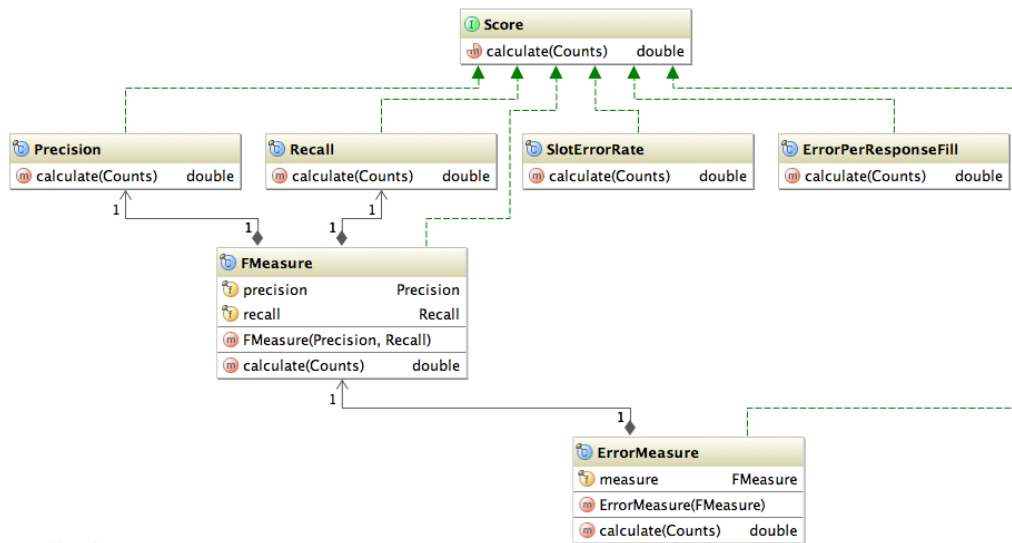
Figure 25: Counter implementation

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 ( $\rho$ )	Precision
Recall ( $\pi$ )	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 26:** Score implementation

Listing 15 shows an example **Score** implementation, in this case **Recall**.

```

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 15: Recall score implementation

### 6.6.3 Performance evaluation

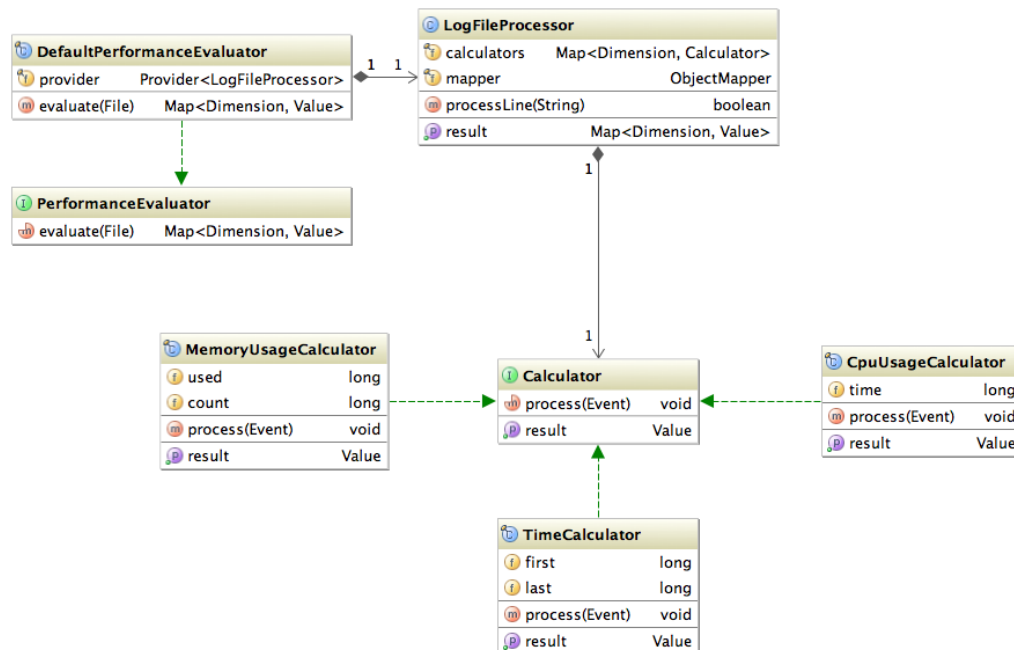


Figure 27: PerformanceEvaluator implementation

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 16:** `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 17:** `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 18 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 18:** `MemoryUsageCalculator`

## 6.7 Implementation obstacles

The next chapter will describe and discuss the obstacles that occurred during the implementation and testing phase of the framework and how they problems that arised got solved.

### 6.7.1 Persistence

Banshie’s persistence layer is based on the 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* [54]. A persistence bundle is a bundle with the following bundle header:

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

**Listing 19:** 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*.

## 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 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 exemplary 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<sup>1</sup> 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<sup>2</sup> 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. [3]

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

---

<sup>1</sup><http://nlp.stanford.edu/~manning/>

<sup>2</sup><http://opennlp.apache.org/>

### 7.1.2 Stanford CoreNLP

Stanford CoreNLP <sup>1</sup> is an integrated framework, which makes it very easy to apply a bunch of language analysis tools to a piece of text. Stanford CoreNLP integrates tools, like a part-of-speech tagger, a named entity recognizer, a parser, a coreference resolution system, and provides model files for analysis of english text. 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. [65]

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

## 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 6.4. For the sake of readability of this document and to keep the evaluation process understandable an extremely small test document, shown in listing 20, 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 20:** Example document

The example reference output, listing 21, 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.

---

<sup>1</sup><http://nlp.stanford.edu/software/corenlp.shtml>

```

<?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 21:** Example extraction reference

### 7.2.1 Apache OpenNLP

Listing 22 shows the most relevant part of the adapter implementation for the Apache OpenNLP Name Finder Tool. The document reading and output writing part of the code was omitted to improve readability. 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 then passed on to the `NameFinder` which finds spans and marks them as persons, organizations or locations accordingly.

```

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() - 1].getEnd();

        // ...
    }
}

```

**Listing 22:** 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 needed.

Listing 23 shows the extraction result produced by the adapter implementation.

```

<?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 23:** 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 the 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 24:** Apache OpenNLP evaluation result

The discrepancy between CPU and execution time is based on the fact that there is a small delay between when the process starts and when the first event polling occurs, because the JMX socket connection needs to be established first. Since the process CPU time is determined by the operating system and the total execution time by the difference between the first and the last event there is the possibility of a small gap.

As shown in listing 24, OpenNLP only required about 2.25 seconds CPU time and 35 megabytes of memory to perform the extraction but on the other hand 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 around a pipeline. 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 25.

```
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 25:** Stanford CoreNLP extractor adapter

The extraction of the CoreNLP adapter is shown in listing 26 and it's obvious that it contains much more annotated tokens than the OpenNLP result in listing 23. But since the Stanford NER tagger works on tokens, it tags individual words with NER types.

```
<?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 26:** Stanford CoreNLP extraction result

Listing 27 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 27:** Stanford CoreNLP evaluation result

### 7.3 Comparison

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	$t_{cpu}$	$m$	$t$	$\pi$	$\rho$	$F$	$E$	$ERR$	$SER$
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 comparison. Even though the example document and the 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.

Because the example used above is tiny and therefore not very realistic we will run two extraction evaluations with a subset of the Enron Email Dataset <sup>1</sup>. The subset contains approximately 3,000 email messages with about 400,000

<sup>1</sup><http://www.cs.cmu.edu/~enron/>

words total. Since the dataset just contains the raw data and no NER reference set, we will run both extractors against each other. The results of this direct comparison is shown in the following tables.

$t_{cpu}$	$m$	$t$	$\pi$	$\rho$	$F$	$E$	$ERR$	$SER$
83070.0	171.0	78999.0	0.893	0.260	0.403	0.561	0.747	0.770

**Table 9:** Apache OpenNLP vs. Stanford CoreNLP

$t_{cpu}$	$m$	$t$	$\pi$	$\rho$	$F$	$E$	$ERR$	$SER$
883330.0	1005.0	853000.0	0.339	0.893	0.491	0.508	0.673	1.845

**Table 10:** Stanford CoreNLP vs. Apache OpenNLP

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 runs, 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.

## 7.4 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 22 illustrated.

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 summarizes the work with a review and discusses some lessons learned in the process. It also gives an outlook for future work and research that can be done based on the results of this thesis.

### 8.1 Review

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

Additionally I gave an overview over the current state of IE , the most common IE methods and approaches as well as an introduction to IE evaluation methodology. We discussed different matching rules, mapping algorithms and performance and runtime performance measures for evaluating information extraction systems.

The delivered framework prototype currently supports the *All Occurences* IE approach and mixed rule approach (,contains‘ and ,overlap‘) for matching reference answers and predictions. It includes, but is not limited to, six different performance (precision, recall, F-Measure, Error measure, Error per response fill and slot error rate) and two runtime performance measures (CPU time and memory consumption). The current version of the software is limited to JVM-based extractors, due to the fact that the collecting of the runtime performance measures is realized using JMX tools.

The platform was planned to include a web based user interface which provides HTTP uploads for IE programs, monitoring execution progress as well as statistics and scores in form of diagrams. Unfortunately, due to time management issues, this feature has not been realized.

## 8.2 Lessons learned

Modularity is very common buzzword in software engineering, but as so often it's easier said than done. Writing modular software is not easy. It's an ongoing process which consists of continuous dependency management. One needs to split up and introduce levels of indirection to break up too tightly coupled modules. What even makes writing modular software more difficult is that there is no perfect way for modularity, every decision is a compromise between more coarse-grained, and easier to use, modules and more fine-grained, and easier to reuse, modules. But on the other hand good modular architecture is very similar to good class/package level design, you get used to it and apply patterns without even realizing you are doing so.

Evaluating information extraction systems uniformly is not an easy task either. First of all, very much like the area of IE itself, the process of evaluating is as inexact, because there is no general agreement about correct answers of extraction. The desired results differ from domain to domain and even from use case to use case. There might always be a human factor involved in IE evaluation to define gold standards for extraction results, since it's so highly subjective.

Another reason why it's difficult to evaluate different tools is the need to find a common data format for extractors which, on the one hand, is simple enough to be adapted to but, on the other hand, flexible enough to handle different IE approaches and tasks. Filling templates, see One Best per Document, and annotating tokens in a document, see All Occurrences, are two relatively different approaches and offering a single output format capable of supporting both approaches is not straightforward.

Writing adapters for existing IE tools to run and test them in Banshie is not particularly hard but still necessary. The APIs of the selected extractors were reasonable simple to understand and adapt to the framework's interface specification. But different systems and their respective APIs might be more difficult.

### 8.3 Outlook and future work

Banshie’s architectural design is based on solid, state-of-the-art patterns, but to expand the framework’s capabilities beyond prototype character several 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 Recognition*, 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 [46].

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 in order 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 on 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.

Banshie’s extractor interface specification states, that extractors should read documents from the standard input which forces them to run in single-document mode. Although it’s still possible to feed multiple documents to an extractor at once by just concatenating them, a real multi-document mode for extractors would be preferable. A possible solution would be a separate XML input schema and a more sophisticated output format which allows collections of documents.

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 additional custom command line parameters to the external Java process would be

handy too.

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 straightforward 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 which could be used by alternative *event production* implementations.

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] Alan Akbik et al. „Unsupervised Discovery of Relations and Discriminative Extraction Patterns“. In: *Proceedings of 24th International Conference on Computational Linguistics*. Vol. 2. 2012.
- [2] 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.
- [3] *Apache OpenNLP*. 2010. URL: <http://opennlp.apache.org/>.
- [4] 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>.
- [5] Alan Bateman. *JEP 102: Process API Updates*. 2011. URL: <http://openjdk.java.net/jeps/102>.
- [6] *Bnd*. 2006. URL: <http://www.aqute.biz/Bnd/Bnd>.
- [7] Kai-Uwe Carstensen et al. *Computerlinguistik und Sprachtechnologie: Eine Einführung*. 3. Aufl. Spektrum Akademischer Verlag, 2010. ISBN: 9783827414076.
- [8] Kyle W. Cartmell. *Five Common java.lang.Process Pitfalls*. 2009. URL: <http://kylecartmell.com/?p=9>.
- [9] 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>.
- [10] 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.
- [11] 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.

- [12] 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](http://www.itl.nist.gov/iaui/894.02/related_projects/muc/proceedings/ie_task.html).
- [13] Nancy Chinchor. *What is Information Extraction?* 2001. URL: [http://www.itl.nist.gov/iaui/894.02/related\\_projects/muc/info/w\\_hats\\_ie.html](http://www.itl.nist.gov/iaui/894.02/related_projects/muc/info/w_hats_ie.html).
- [14] 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>.
- [15] Michael Crystal et al. „A methodology for extrinsically evaluating information extraction performance“. In: *In Proc. of HLT/EMNLP*. 2005, pp. 652–659.
- [16] Hamish Cunningham. „Information Extraction, Automatic“. In: *Encyclopedia of Language and Linguistics* 2 (2005).
- [17] 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.
- [18] Hamish Cunningham et al. *Text Processing with GATE (Version 6)*. 2011. ISBN: 978-0956599315. URL: <http://tinyurl.com/gatebook>.
- [19] 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.
- [20] A. Douthat. „The Message Understanding Conference Scoring Software User’s Manual“. In: 1998.
- [21] Line Eikvil. *Information Extraction from World Wide Web - A Survey*. 1999.
- [22] *Ellogon*. URL: <http://www.ellogon.org/>.



- [23] 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>.
- [24] 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.
- [25] Martin Fowler. *Inversion of Control Containers and the Dependency Injection pattern*. 2004. URL: <http://www.martinfowler.com/articles/injection.html>.
- [26] Dayne Freitag. „Machine Learning for Information Extraction in Informal Domains“. PhD thesis. Computer Science Department, Carnegie Mellon University, 1998.
- [27] Dayne Freitag and Nicholas Kushmerick. „Boosted Wrapper Induction“. In: AAAI Press, 2000, pp. 577–583.
- [28] Robert Gaizauskas and Yorick Wilks. *Information Extraction: Beyond Document Retrieval*. 1998.
- [29] *GATE Information Extraction*. URL: <http://gate.ac.uk/ie/>.
- [30] Andrew Goldberg. „Advanced NLP: Automatic Summarization“. 2007.
- [31] *Google Guice*. 2012. URL: <https://code.google.com/p/google-guice/>.
- [32] R. Grishman. „Discovery methods for information extraction“. In: *ISCA & IEEE Workshop on Spontaneous Speech Processing and Recognition*. 2003.
- [33] R. Grishman. „NLP: An information extraction perspective“. In: *AMSTERDAM STUDIES IN THE THEORY AND HISTORY OF LINGUISTIC SCIENCE SERIES 4* 292 (2007), p. 17.

- [34] 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](http://dx.doi.org/10.1007/3-540-63438-X_2).
- [35] 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>.
- [36] *Interface OperatingSystemMXBean*. 2013. URL: [http://docs.oracle.com/javase/6/docs/jre/api/management/extension/com/sun/management/OperatingSystemMXBean.html#getProcessCpuTime\(\)](http://docs.oracle.com/javase/6/docs/jre/api/management/extension/com/sun/management/OperatingSystemMXBean.html#getProcessCpuTime()).
- [37] Panos Ipeirotis. *Evaluating Information Extraction using xROC Curves*. 2007. URL: <http://www.behind-the-enemy-lines.com/2007/07/evaluating-information-extraction-using.html>.
- [38] 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.
- [39] *JSR 330: Dependency Injection for Java*. 2009. URL: <http://jcp.org/en/jsr/detail?id=330>.
- [40] Kirk Knoernschild. *Java Application Architecture: Modularity Patterns with Examples Using OSGi (Robert C. Martin Series)*. 1st ed. Prentice Hall, Mar. 2012. ISBN: 9780321247131.
- [41] In Knorz et al. *Automatic Document Classification: A thorough Evaluation of various Methods*. 2000.
- [42] Andrew Lampert. *A Quick Introduction to Question Answering*. 2004.
- [43] 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.
- [44] W. Lehnert et al. *Evaluating an Information Extraction System*. 1994.

- [45] Linsmayr. „Evaliex - Information Extraction Evaluation Framework“. Master thesis. Nov. 2010.
- [46] Elisabeth Linsmayr. „GeMap Generic Mapper for XML Instances“. Softwaredokumentation Version 2. 2010.
- [47] 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.
- [48] John Makhoul et al. „Performance Measures For Information Extraction“. In: *In Proceedings of DARPA Broadcast News Workshop*. 1999, pp. 249–252.
- [49] Robert C. Martin. *Agile Software Development, Principles, Patterns, and Practices*. 1st. Prentice Hall, Oct. 2002. ISBN: 9780135974445. URL: <http://amazon.com/o/ASIN/0135974445/>.
- [50] Diana Maynard et al. „Ontology-based information extraction for market monitoring and technology watch“. In: *In ESWC Workshop "End User Aspects of the Semantic Web"*. 2005.
- [51] Günter Neumann. „Informationsextraktion“. In: *Computerlinguistik und Sprachtechnologie - Eine Einführung* (2001).
- [52] 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>.
- [53] *OSGi Alliance Specifications*. URL: <http://www.osgi.org/Specifications/HomePage> (visited on 07/10/2012).
- [54] *OSGi Service Platform Release 4 Version 4.2 Enterprise Specification*. URL: <http://www.osgi.org/Download/Release4V42>.
- [55] Thierry Poibeau and Cédric Messiant. *Do we still Need Gold Standards for Evaluation?* 2008.
- [56] Gunnar Rätsch. *A Brief Introduction into Machine Learning*. 2004.
- [57] C.J. van Rijsbergen and Ph. D. *Information Retrieval*. 1979.

- [58] 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>.
- [59] Yutaka Sasaki. *The truth of the F-measure*. 2007.
- [60] Marcus Schramm. „Informationsextraktion auf Basis strukturierter Daten“. MA thesis. TU Dresden, 2008.
- [61] 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.
- [62] 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.
- [63] An De Sitter, Tood Calders, and Walter Daelemans. „A Formal Framework for Evaluation of Information Extraction“. In: *A3* (Apr. 2004).
- [64] Mark Stamp. *Once Upon a Time-Memory Tradeoff*. 2003.
- [65] *Stanford CoreNLP*. 2012. URL: <http://nlp.stanford.edu/software/corenlp.shtml>.
- [66] 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>.
- [67] J. Weinhofer. „Extraktion semantisch relevanter Daten aus natürlich sprachlichen Inhalten in Hinblick auf eine automatische Fragengenerierung“. MA thesis. Technische Universität Graz, 2010.
- [68] Wikipedia. *Information Extraction*. 2012. URL: [http://en.wikipedia.org/wiki/Information\\_extraction](http://en.wikipedia.org/wiki/Information_extraction) (visited on 01/19/2013).
- [69] Wikipedia. *Modularity*. 2012. URL: [http://en.wikipedia.org/wiki/Modularity#Modularity\\_in\\_technology\\_and\\_management](http://en.wikipedia.org/wiki/Modularity#Modularity_in_technology_and_management).
- [70] Wikipedia. *Precision and recall*. 2012. URL: [http://en.wikipedia.org/wiki/Precision\\_and\\_recall](http://en.wikipedia.org/wiki/Precision_and_recall).

- [71] 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](http://dx.doi.org/10.1007/3-540-63438-X_1).
- [72] Osmar R. Zaïane. *Introduction to Data Mining*. 1999.

## List of Figures

1	Annotated example document for the AO approach . . . . .	18
2	Precision formula . . . . .	20
3	Recall formula . . . . .	21
4	Recall and precision example figure [70] . . . . .	21
5	F-measure formula . . . . .	22
6	$F_\beta$ -score formula [11] . . . . .	22
7	$F_\alpha$ -score formula [48] . . . . .	22
8	Error measure formula . . . . .	23
9	Error per response fill formula . . . . .	23
10	Slot Error Rate formula . . . . .	23
11	Error measure for $\alpha = 0.5$ . . . . .	24
12	Rewritten error per response fill formula . . . . .	24
13	Module definition diagram . . . . .	28
14	OSGi layered model [53] . . . . .	29
15	Module dependencies . . . . .	41
16	Banshie model and persistence API . . . . .	43
17	Banshie Execution API . . . . .	44
18	Banshie Evaluation API . . . . .	45
19	Engine implementation . . . . .	50
20	ProcessService implementation . . . . .	52
21	ProcessMonitor implementation . . . . .	53
22	EventProducer implementation . . . . .	55
23	EventLogger implementation . . . . .	57
24	QualityEvaluator implementation . . . . .	58
25	Counter implementation . . . . .	58
26	Score implementation . . . . .	59
27	PerformanceEvaluator implementation . . . . .	60

## List of Tables

1	Message Understanding Conferences . . . . .	6
2	Named Entity Recognition example output . . . . .	8
3	Template Element Construction example output . . . . .	9
4	Template Element Construction example output . . . . .	9
5	Example template for the OBD approach . . . . .	17
6	Confusion matrix . . . . .	19
7	Metric implementations . . . . .	59
8	Evaluation result comparison . . . . .	72
9	Apache OpenNLP vs. Stanford CoreNLP . . . . .	73
10	Stanford CoreNLP vs. Apache OpenNLP . . . . .	73

## List of Listings

1	Guice module . . . . .	32
2	Constructor injection . . . . .	33
3	Peaberry lifecycle annotation . . . . .	34
4	Peaberry bundle header . . . . .	34
5	Maven Bundle Plugin usage . . . . .	35
6	Banshie API usage . . . . .	46
7	Extractor manifest header . . . . .	47
8	Banshie XML Schema . . . . .	47
9	Banshie XML Example . . . . .	48
10	ProcessService API example usage . . . . .	51
11	Scheduling in java . . . . .	53
12	Configuring process to use JMX . . . . .	54
13	JMX connection . . . . .	54
14	Example event log file excerpt . . . . .	57
15	Recall score implementation . . . . .	60
16	LogFileProcessor . . . . .	61
17	Calculator Interface . . . . .	61
18	MemoryUsageCalculator . . . . .	62
19	Persistence bundle header . . . . .	63
20	Example document . . . . .	66
21	Example extraction reference . . . . .	67
22	Apache OpenNLP extractor adapter . . . . .	68
23	Apache OpenNLP extraction result . . . . .	69
24	Apache OpenNLP evaluation result . . . . .	69
25	Stanford CoreNLP extractor adapter . . . . .	70
26	Stanford CoreNLP extraction result . . . . .	71
27	Stanford CoreNLP evaluation result . . . . .	72