Institute for Natural Language Processing

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diploma Thesis No. 3569

Interactive Exploration and Model Analysis for Coreference Annotations

Markus Gärtner

Course of Study:	Computer Science
Examiner:	Prof. Dr. Jonas Kuhn
Supervisor:	M.Sc. Anders Björkelund
Commenced:	April 19, 2013
Completed:	September 27, 2013
CR-Classification:	D.1.5 H.5.2 J.5

Abstract

I present the design and implementation of an interactive visualization- and exploration-framework for coreference annotations. It is designed to meet the needs of multiple different users on a modern and multifaceted graphical exploration tool. To demonstrate its suitability for these various needs I outline several use cases and how the framework can help users in their individual tasks.

It offers the user different views on the data with additional functionality to compare several annotations. Complex analysis of annotated corpora is supported by means of a search engine which lets the user construct queries both in a graphical and textual form. Both qualitative and quantitative result breakdowns are available and the implementation features specialized visualizations to aggregate complex search results. The framework is extensible in many ways and can be customized to handle additional data formats.

Contents

List of Figures

List of Tables

1 Introduction

The linguistic term *coreference* denotes that two or more expressions in a text represent or refer to the same entity. These so called *mentions* of a mutual *referent* (the entity they all refer to) can be distributed across several sentences in a text, making coreference an inter-sentential relation. The text below is an excerpt from a larger document and illustrates an example for coreference structures, showing the mentions of four entities marked with brackets:

> In the summer of 2005, a picture that people have long been looking forward to started emerging with frequency in various major [Hong Kong]$_{a_1}$ media. With [their]$_{b_1}$ unique charm, [these well-known cartoon images]$_{b_2}$ once again caused [Hong Kong]$_{a_2}$ to be a focus of worldwide attention. [The world's fifth [Disney]$_{d_1}$ park]$_{c_1}$ will soon open to the public here. The most important thing about [Disney]$_{d_2}$ is that [it]$_{d_3}$ is a global brand. Well, for several years, although [it]$_{c_2}$ was still under construction and, er, not yet open, it can be said that many people have viewed [Hong Kong]$_{a_3}$ with new respect. Then welcome to the official writing ceremony of [Hong Kong Disneyland]$_{c_3}$.

Figure 1.1: An example text with mentions from four sets being marked. The sentences are the beginning of a larger document from the CoNLL-2012 Shared Task development set (Pradhan et al., 2012).

To make the text more readable only those mentions are marked that are part of an actual coreference relation. Mentions indexed by the same character refer to the same real world entity and are therefore coreferent. For example the mentions $\{c_1,c_2,c_3\}$ all refer to the park *"Hong Kong Disneyland"*.

Besides having a common *referent* entity mentions can also refer to one another. Looking at the same set of mentions it becomes obvious that *"it"*$_{c_2}$ directly refers back to the previous mention *"The world's fifth Disney park"*$_{c_1}$. On the other hand the mentions $_{b_1}$ and $_{b_2}$ form a forward reference where the pronoun *"their"*$_{b_1}$ refers to the later introduced phrase *"these well-known cartoon images"*$_{b_2}$. These linguistic phenomena are called *anaphora* for backwards references and *cataphora* for references to mentions that appear later in a text.

The task of deciding whether two mentions are coreferent is known as *coreference resolution*. It is an important subtask in natural language processing (NLP) systems such as information extraction and discourse analysis. Even for human readers the task of resolving a coreference relation is often non-trivial. The knowledge one requires to solve this task spans over multiple levels, from morphological and lexical information all up to semantics (Mitkov, 2003, pp. 266-279). Several resolver systems have been developed (Fernandes et al., 2012; Björkelund and Farkas, 2012; Soon et al., 2001) and a number of projects for annotating corpora with coreference information exist (Eckart et al., 2012; Poláková et al., 2012; Pradhan et al., 2007).

Existing visualization tools dealing with coreference annotated data were often developed as annotation tools in the first place. This way they primarily serve the purpose of assisting in an annotation task and not in providing special visualization and exploration capabilities. Considering the importance of coreference resolution, there is a strong need for dedicated tools which can properly handle the specific properties of coreference structures.

In this thesis I will present the design and implementation of a very flexible coreference visualization and exploration framework suitable for a broad audience. For the course of this thesis it will go by the title of IEC which is short for *"Interactive Exploration tool for Coreference data"*. It attempts to satisfy the needs for users that motivated by their background are interested in rather different aspects of coreference and therefore require specialized tools. It features advanced implementations of three different views on coreference and a data model that aims not to restrict the variety of possible input data. Altogether it is designed to provide a maximum of usability. The integrated search engine supports very expressive (graphical) queries and has the ability to create both qualitative and quantitative breakdowns of results. This renders it very practical for complex exploration and analysis of coreference models.

The remainder of my thesis is structured as follows: Chapter 2 elaborates the motivation for implementing a new framework and gives a more in-depth background on the topic. In Chapter 3 I list a selection of existing tools and projects that aim on providing similar software. Chapter 4 outlines the features of IEC and their functionality while Chapter 5 presents a more technical overview of the architecture. Chapter 6 evaluates in which ways the implementation can be applied to real data and how it meets the requirements of different user groups and Chapter 7 concludes.

2 Background and Motivation

The phenomenon of coreference plays an important role in several tasks in the field of (computational) linguistics. It is therefore subject to examination from a wide range of disciplines. The people which are interested in coreference often focus on different aspects of it and for this reason require a special *view* on the data they are working with. This chapter is intended to give a wider background overview on the field of coreference and the various parties working with it and to emphasize the demand for new tool frameworks that specifically address the needs of those parties. The first sections explain the possible ways coreference can be represented and visualized. This not only includes well established designs but also new ideas. In Section 2.3, I will outline the different groups of users and their individual requirements regarding a visualization and exploration framework for coreference. The final Section 2.4 summarizes the chapter and provides a basic overview on the design idea of my framework.

2.1 Forms of Coreference Representation

The most basic way of representing a coreference structure is to picture it as a collection of distinct *sets* of mentions. Each set or *cluster* encapsulates all the mentions that are considered coreferent. An example for this representation being used is the CoNLL-2012 Shared Task (Pradhan et al., 2012) format.[1] A clear disadvantage of this representation is that it is very flat and does not provide real structural information: It does not honor phenomena like anaphora or cataphora or inter-mention relations at all and instead treats all mentions in a cluster alike. The often used *chained* model of a cluster only provides the benefit of preserving the order of mentions as given by the raw text.

As an alternative, a tree-based model for coreference has been proposed (Fernandes et al., 2012) which is able to encode a much richer set of features. Figure 2.1 shows the basic difference between a *flat* chained structure and the tree representation. Considering modern approaches to coreference resolution (Soon et al., 2001) that are based on the mention-pair model (Ng, 2010) trees can be seen as sort of a natural representation. The idea of this model is for a pair of mentions to decide whether or not they are coreferent or to provide a probability. This can be done by means of enforcing similarities between mentions in terms of features like gender, number, etc. Traversing the mentions in a given text in reading order the system then has to find the most probable already visited candidate to form a coreference relation. Since this

[1] `http://conll.cemantix.org/2012/data.html`

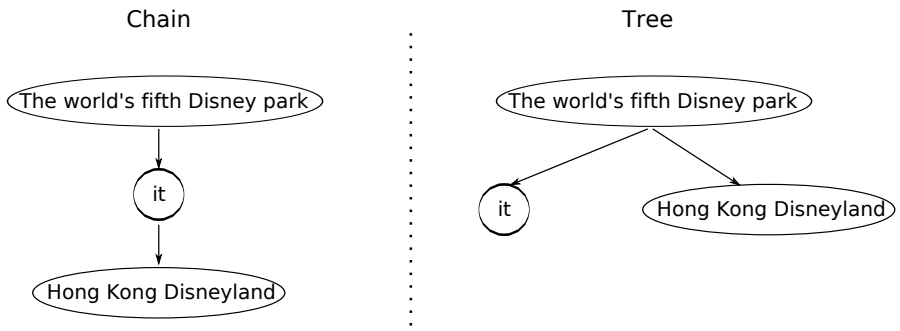


Figure 2.1: Two representations of coreference structures as generated by coreference resolution tools. The mentions are members of the set $\{c_1,c_2,c_3\}$ from the example text in Figure 1.1.

automatically forms directed links between mentions the overall structure of a cluster becomes a list (if all mentions refer back to their nearest cluster neighbor) or a tree. It is important to note that the tree representation allows for more fine-grained comparison or error analysis of the output generated by automatic coreference resolver systems.

Besides the structural differences between models, the total amount of additional information being encoded varies greatly. Annotation projects or automatic coreference systems all store features in their models which are specific to their individual task. This makes it particularly difficult for visualization or exploration tools to properly handle all the various models.

2.2 Visualization of Coreference Structures

> In the summer of 2005, a picture that people have long been looking forward to started emerging with frequency in various major [Hong Kong]$_{a_1}$ media. With [their]$_{b_1}$ unique charm, [these well-known cartoon images]$_{b_2}$ once again caused [Hong Kong]$_{a_2}$ to be a focus of worldwide attention. [The world's fifth [Disney]$_{d_1}$ park]$_{c_1}$ will soon open to the public here. The most important thing about [Disney]$_{d_2}$ is that [it]$_{d_3}$ is a global brand. Well, for several years, although [it]$_{c_2}$ was still under construction and, er, not yet open, it can be said that many people have viewed [Hong Kong]$_{a_3}$ with new respect. Then welcome to the official writing ceremony of [Hong Kong Disneyland]$_{c_3}$.

Figure 2.2: The example text from Figure 1.1 with cluster-based color highlighting.

Regardless of the exact format there will always be the need to visualize and explore coreference annotations in an interactive way. Tools presenting coreference structures and their underlying text typically generate **floating text** and visually highlight relevant portions or surround them with square brackets. Several examples for this visualization technique are listed and described in Chapter 3. Figure 2.2 shows the annotated text from the introduction enriched by colored highlighting of the mentions based on their cluster. This visualization presents a very intuitive and from a readers perspective easy to understand way of outlining clusters over a large portion of text. It is however unable to properly handle structural information like

links between mentions. Attempting to graphically link mentions across floating text usually generates very confusing results, and is therefore not a viable solution.

When one is interested in the mere structural properties of a coreference annotation it is possible that the parts of text not affected by the annotation are of minor interest. In this case visualizing the structure as a **graphical tree** provides a viable solution. For this approach mentions are transformed to nodes and the links between them translate into directed edges. Figure 2.1 in the previous section shows an example for visualizing a cluster as a tree. Compared to the text highlighting mentioned above, this technique provides a purely structure-focused view on the data. In addition to the very simple version in the figure it is also possible to include additional information in the nodes and edges of the tree to further improve the visualization.

Sentence	a	b	c	d
1	X			
2	O	S		
3			S	X
4				S
5	O		S	
6			X	

Table 2.1: Visualization of the coreference structure introduced in Figure 1.1 in the form of an *entity grid*. The value in a cell shows which syntactical function a given entity represented by the cluster $x\epsilon\{a, b, c, d\}$ has in a particular sentence. The values are `S` for *subject*, `O` for *object* and `X` for *other*.

As a third way of visualizing coreference structures I propose a tabular view that is inspired by the so called *Entity Grid*. The entity grid was originally introduced to model local coherence (Barzilay and Lapata, 2008) of texts or discourses, i.e., the way speakers or writers make transitions across sentences "smooth". It is essentially a table that lists sentences as rows and entities as columns. A cell indicates the syntactic function an entity has in a given sentence. The possible values for a cell (provided the entity occurs in a given sentence) are `S` for *subject*, `O` for *object* and `X` for *other*. Table 2.1 shows the example text from the introduction as it would appear in an entity grid. Note that however the basic specification of the entity grid exists for some time there are to my knowledge still no implementations of tools featuring a graphical interface based on that grid view.

My idea is to take the concept of a tabular outline and apply it to coreference annotations while lifting the limitations on what has to be the label for a given cell. Again a row in the table represents a sentence and each column holds a single cluster with all the mentions of that cluster's entity. The value of a cell should not be restricted but be left to the **user's decision**. The compact nature of the grid is well suited to provide a brief overview, even on an entire document. Combined with the flexibility regarding the actual content of a grid cell I propose, this design can help users to get a valuable view on entities in a document and how their references behave.

2.3 User Groups

Very tightly coupled with the actual view on coreference structures is the context in which it is to be used and by whom. For the course of this thesis I will focus on two types of potential users as representatives for groups with rather different expectations and needs. I am going to outline for both groups the aspects of coreference they might be interested in and what requirements a tool should meet in order to assist them.

The first group labeled as **corpus linguists** will be focused on analyzing a gold or automatically annotated corpus or set of corpora for special phenomena. Their questions for a search engine would be of the form *"Is this construct a rare phenomenon? If yes, how rare exactly?"* or *"How often does this phenomenon A appear while phenomenon B is absent? What are actual instances of A in my corpus?"*. To answer such questions they need interfaces to query a corpus for potentially very complex structures and require both qualitative and quantitative breakdowns of the results. In this context I think the most vital visualizations will be the text outline and entity grid. The first to view coreference annotations in their textual context and the latter in order to follow several entities over the course of a document and compare their role or other properties. For non-technical persons it is also of importance that the tools are easy to use and do not require deeper technical knowledge than the linguistic task itself.

In the second group I collect all **developers** of coreference related systems or other NLP software. Those people will be primarily interested in the investigation of critical phenomena that can cause problems or difficulties to their systems. For example the developer of a coreference resolver when looking at the predicted output of his tool could ask *"Given a clear case of mismatch, how often does it happen? In which context does it happen?"* or *"On which node in the tree does my resolver make its first mistake?"*. This indicates that unlike the regular corpus linguist a developer usually requires a more structure focused visualization. Here the tree-outline and entity grid dominate over the pure textual representation in terms of importance. Furthermore this type of user has the possible need for an additional visualization feature not mentioned before: The ability to directly compare two annotations for the same base data. To properly evaluate the predictions of an automated system it is often vital to perform a very fine-grained comparison between both the gold data and the predicted output. This should be supported by an interactive tool that wishes to assist NLP developers. In addition it would be a helpful feature to be able to systematically search for differences in those two annotations.

2.4 Summary

In the previous sections of this chapter I have outlined several properties of coreference annotations that, when combined, pose a strong requirement for a dedicated visualization and exploration framework to be developed:

- Different user groups focus on very different aspects of coreference and therefore need highly specialized tools to assist them.

- The trend in representing coreference structures leads to tree-based models. New tools should be able to handle both unstructured *"old"* data as well as the relatively new models.
- There are multiple ways to visualize coreference structures. Besides the more *"classic"* approaches I have presented the design of a tabular view resembling an improved entity grid.

As will be shown in Chapter 3 several tools exist which address certain of the needs stated above. However most of those tools were designed as annotation tools and in most cases not even specifically to handle coreference. As a result they serve their intended role in assisting human annotators but lack the flexibility to perform specialized visualization and exploration tasks on coreference annotations. In addition my suggested tabular visualization approach has not been implemented before in any way.

IEC was designed to satisfy as many of the above requirements as possible. It is dedicated to the tasks of visualization and exploration of coreference annotations. For both tasks it features special tools with a maximum of usability. In terms of visualization it offers implementations of all three techniques listed above. Additionally it is able to provide comparative views for two different annotations. To support a wide range of coreference annotations, its internal data format is minimalistic in its core but very extensible. Using the plugin-based architecture anybody can also include a custom format for new annotations.

3 Related Work

Coreference annotations in their role as an important subtask of NLP have steadily grown in terms of received attention. In contrast, the number of tools developed to visualize and explore said annotations is still quite low. In this chapter I will introduce a selection of (annotation) tools that can be used to inspect coreference annotations and perform search operations of varying complexity on them.

3.1 MMAX2

MMAX2 is an annotation tool (Müller and Strube, 2000, 2006) that offers various ways to interact and modify annotated corpora. It is highly customizable both in terms of the data formats it supports and the visualization capabilities on the user front-end. Although it is not specifically designed for visualization of coreference structures, the flexible nature of its data model enables it to process them. It relies on the principles of *stand-off annotation* (Thompson and McKelvie, 1997), where annotations should not modify the underlying data in a corpus and preferably be entirely separated from it, and *multi-level annotation*. Stand-off annotation allows several annotations representing different phenomena to be contained in the same corpus, while the principle of multi-level annotation enables phenomena on different levels to be related to each other.

MMAX2 uses XML files for both the specification of possible annotation by means of an annotation scheme file and as storage for *base data* (the raw corpus data) and annotation data. Its main purpose as an annotation tool is to support human annotators in the process of an annotation task. Besides this core functionality it also features several useful tools to explore and otherwise manipulate a corpus.

The most relevant functionality in the context of this thesis is the querying tool that lets the user search a corpus for examples of certain phenomena. It is based on a multi-level query language called MMAXQL which allows for textual definition of a query. Both the base data and mentions of every available level can be queried with the support of regular expressions and several structural relations regarding the spans of mentions declared in the query. Visualization of corpus elements (sentences) and their annotations in MMAX2 is realized as *highlighted text* where customizable style-sheets are used to create visual enhancements of the plain text comprised of word tokens. These enhancements can be, among others, changes in font type or style, underlining mentions or surrounding them with square brackets or connection lines representing relations between mentions. There is no additional way of visualizing certain structures like trees available in MAAX2 other than the relation lines across floating text or recursive bracketing of spans.

3.2 SALTO

Another graphical tool for manual annotation is SALTO (Burchardt et al., 2006). While other tools similar to MMAX2 perform visualization and annotation on a mere textual representation of the data, SALTO works with graphical tree structures. Another relevant point is the level on which the manual annotation takes place. SALTO works on top of syntactically annotated data and adds a *second structural layer* on top of it. This makes it well suited for the annotation of semantic roles and semantic classes, discourse structure or anaphoric relations. It is however restricted to the scope of a single sentence. Data has to be available in the TIGER XML format (Brants et al., 2002) or the derived SALSA/TIGER XML (Erk and Padó, 2004) which is SALTO's output format. By interfacing with a treebank search engine called TIGERSearch (Lezius, 2002) the tool allows the user to select datasets for annotation by means of queries. Current utilization in the field of coreference annotation includes the DIRNDL (Eckart et al., 2012) project, a discourse corpus of radio news, featuring a very rich annotation scheme (Baumann and Riester, 2012).

3.3 TrEd

Another annotation tool focused on visualizing data in tree form is TrEd (Pajas and Štěpánek, 2009). It is part of a larger framework developed around a generic and open encoding scheme for annotations called PML (Hana and Štěpánek, 2012). Much like MMAX2 it allows users to define scheme files for arbitrary annotations structures and is therefore open to a wide range of existing formats. Once accessible to TrEd, a corpus resource can be queried via its own query language PML-TQ (Pajas and Štěpánek, 2008). A graphical query editor built into TrEd supports the definition of queries both graphically and textually. Besides the basic search functionality, it features reporting capabilities that can present the user with occurrence-counts or distribution information. Depending on the search, query results can be presented by a text report or as a list of matches that can be further browsed.

3.4 BRAT

Unlike the previously introduced tools the brat rapid annotation tool (BRAT) (Stenetorp et al., 2012) is purely web-based. A server side installation of the tool can be accessed from any modern web browser. Visualization is performed by an open-source text annotation visualizer STAV (Stenetorp et al., 2011). It uses a text-based approach and renders annotations and connections between them within the line spacing area, enlarging it when necessary. The highly configurable design of the tool makes it applicable for many annotation structures including coreference. Also implemented by BRAT is a comprehensive search functionality that allows the user to search document collections. Possible constraints include, but are not limited to, text span annotations, relations, or text forms and can be defined via a compact dialog.

To address some of the technical shortcomings of BRAT and to integrate it with a framework for collaborative annotations, a project named WebAnno (Yimam et al., 2013) has been built around it. WebAnno is a web-based annotation tool that, in addition to the original BRAT features, offers project and user management and is able to interface with a crowdsourcing platform. Unlike BRAT it is able to support arbitrarily large documents while still maintaining a sufficiently high performance. It is on the other hand currently limited to only a handful of annotation types which include coreference.

3.5 Summary

I have listed a small collection of current tools which offer a combination of capabilities relevant to my thesis. First comes the visualization of coreference structures in an intuitive and easy to read form. Second is the possibility for a user to explore a given annotated corpus by means of querying. All introduced tools were intrinsically developed as annotation tools or platforms, and in general not focused on coreference data. The lack of dedicated exploration tools for coreference annotations leaves as only alternative the usage of general purpose tools or such that by their extensibility are able to handle them. In consequence of their original purpose (assisting a human user in annotating raw or preprocessed data) they all feature search capabilities with varying complexity and expressiveness to let the user explore and analyze the data he is working with.

In terms of visualization there are only two types available, namely *textual* representation as implemented in BRAT and MMAX2 or a *tree* outline with SALTO or TrEd as representatives. The latter is in addition often restricted to a single sentence and therefore unable to visualize coreference structures spanning an entire document. This is no surprise considering that those tools are not focused on coreference. Usually a tool provides a single type of visualization, effectively locking the user into one *view* on the data. The third possible visualization technique described in the introduction, the entity grid, is actually only a theoretical model and has not yet been implemented in a proper graphical user interface.

In total the collection of available tools is sufficient for their intended purpose, that is the assistance in annotating corpora. Besides that, they generally lack the specialized visualizations or comparison features as listed in Chapter 2. This clearly emphasizes the need for a dedicated framework designed to specifically address those needs not already satisfied by existing tools.

4 Framework Features

IEC itself is a plugin for ICARUS[1](Gärtner et al., 2013), a Java based open-source platform for tools related to corpus analysis and research. The offered functionality of IEC is essentially divided into two collections of tools. One holds the interfaces to manage the coreference data the user wishes to process and simple tools to inspect this data in a manual way. These tools will be covered in the immediately following section. The other set of tools is integrated into the core search engine of ICARUS and will be outlined in Section 4.2. It basically contains components that customize the existing search engine and provide new types of constraints and supported search targets. Section 4.3 shows how the features of IEC's inspection tools can be employed to comfortably browse a given resource for differences compared to a second annotation. The order of introduction follows the steps a new user would probably perform when using the framework to search for phenomena in a corpus.

4.1 Management and Browsing

The purpose of the *Coreference Perspective* as shown in Figure 4.1 is to concentrate all the tools needed for the task of manually browsing coreference annotations in one place. It holds the tools to register and manage the corpora and annotations that a user intends to work with. Besides that, it provides implementations for all the three essential ways of visualizing coreference structures as outlined in the introduction:

1. Text-Outline
2. Graph-Outline
3. Grid-Outline

Each of these outlines will be introduced in a separate portion of this section with a strong emphasis on their individual features.

4.1.1 Coreference Manager

Users of visualization or browsing tools often find themselves using very large collections of data. Certain tasks might even require the combination of several data sources to be combined,

[1]Interactive platform for Corpus Analysis and Research tools, University of Stuttgart

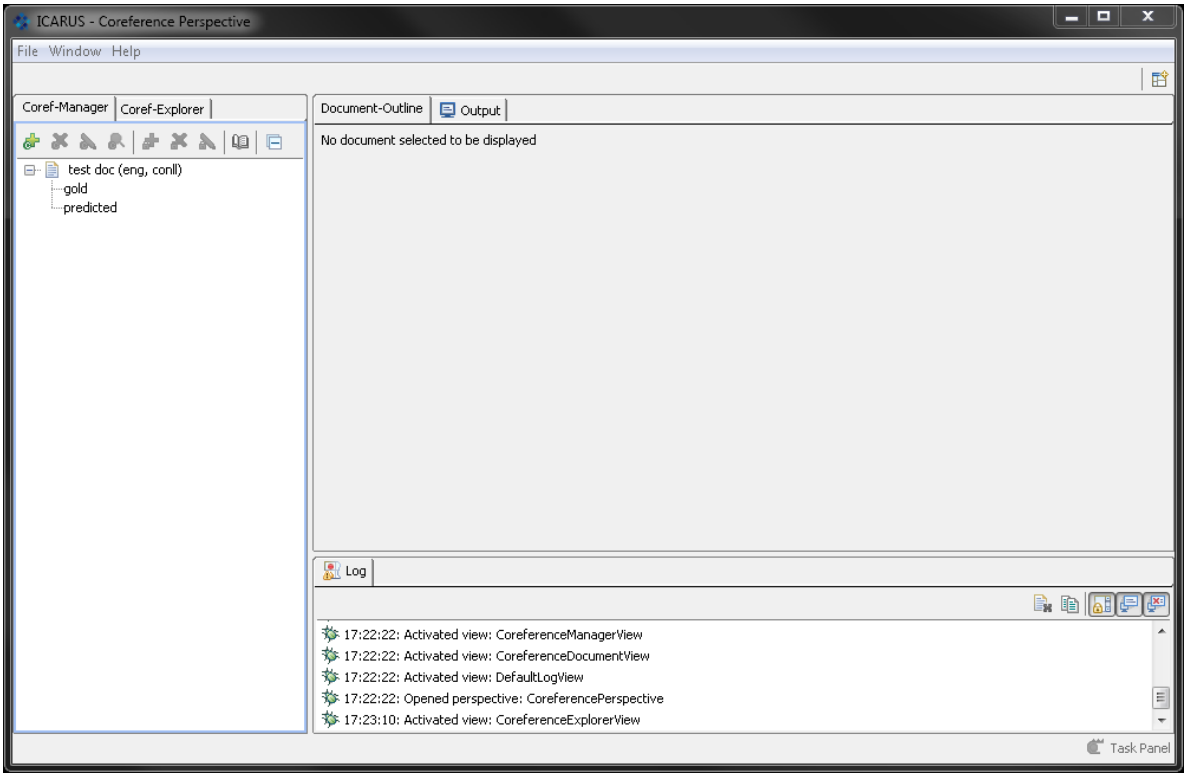


Figure 4.1: Complete screenshot of the *Coreference Perspective* containing interface components to manage document sets, allocations and to inspect them.

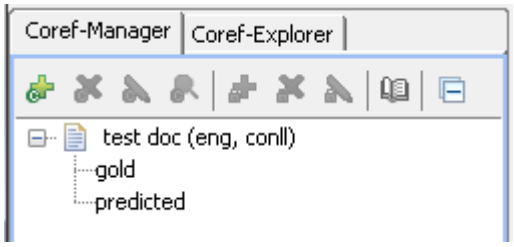


Figure 4.2: *Coreference Manager*-View that allows creation and management of *document sets* and the associated *allocations*.

which in conjunction with the fact that such data collections can be distributed in nature, makes keeping track of them an inherently difficult task for the user.

To aid users in this task, IEC provides a dedicated module called *Coreference Manager*, a snippet of which can be seen in figure 4.2. It allows for the definition of *document sets* that can be used with the various other tools of the platform. A document set is an ordered collection of documents containing an arbitrary number of sentences. Each document represents a contextual boundary for coreference relations (see Section 5.2 for detailed information about the data structures) and together they form a corpus. Each document set is assigned a unique identifier and a pointer to the physical location of the content whereby multiple document sets with different purposes can relate to the same physical data.

IEC reads the format introduced by the CoNLL-2012 Shared Task (Pradhan et al., 2012). The extensibility of the platform being used allows other plugins to contribute new data formats. In addition, the user can define arbitrary textual properties to each document set in the form of key-value pairs that can store further information or meta-data. Multiple so called *allocations*, holding for each document the tree or list structure that represents mentions and their respective coreference relations, can be assigned to each document set. See Appendix A.1 for a detailed format specification. The same statement about extensibility applies to allocations. After the initial set-up of document sets and their allocations in the manager they can be accessed with other tools.

4.1.2 Coreference Explorer

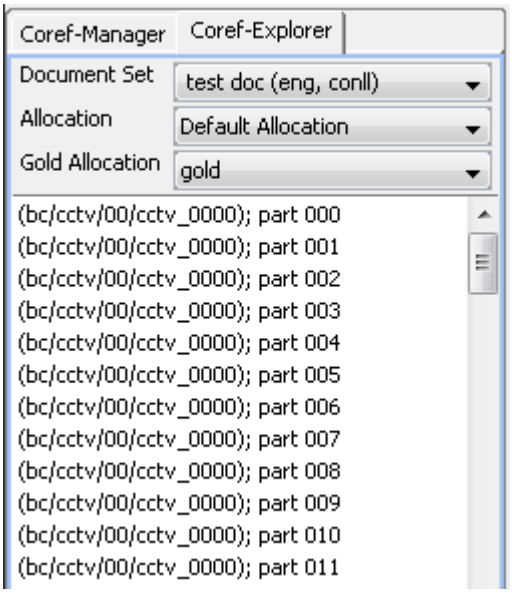


Figure 4.3: *Coreference Explorer*-View showing a list outline of the documents contained in a selected *document set*.

A possible first step in the further use of registered document sets is the simple browsing of the contained data. IEC ships with a special module named *Coreference Explorer* which lets the user select a document set to be inspected. Figure 4.3 shows the user interface of this tool with an example document set currently picked. Note the two additional drop-down menus below the document set selection where the user can assign up to two allocations previously associated with the document set. The term `Default Allocation` in the options refers to the optional allocation that a document set can contain in its original data. As the name suggests an item in the `Gold Allocation` menu will be referred to as gold standard or *correct* version of an annotation whereas the `Allocation` choice signals an allocation to be of predicted nature that needs to be compared against the *correct* one. The individual assignment of allocations in this part of the interface controls how later visualizations present the data to the user.

Selecting a specific document in the list below sends it to the next component of the coreference perspective called the *Document Outline* which manages the actual graphical representation. The document outline will be covered in the next section. Note that the following sections related to the different ways of visualizing the data only assume one allocation assigned by the user and only in Section 4.3 the additional features regarding a second allocation will be explained.

4.1.3 Document Outline

One of the most crucial aspects of interactive systems that deal with abstract data is the way this data is visualized and how the user can customize the visualization engine or otherwise interact with it. Some data structures like for example parse trees in the form of phrase- or dependency-structures 'demand' a certain type of visualization by nature. Parse trees are typically presented in a sort of graph view outlining the tree structure. *In-place visualization*, that is embedding the structural information to be visualized directly into the basic textual form the underlying text comes in, is possible for trees but very difficult for human users to follow when reading. Coreference structures on the other hand can be viewed in many different ways, three of which are implemented in IEC.

Text-Outline

The first and most common type of presenting coreference annotations is a simple **textual** in-place approach as shown in Figure 4.4. All mentions in a document are enclosed by square brackets and highlighted with a certain color where mentions sharing a color belong to the same cluster. Besides the default highlight type of coloring the background of a mention's span, the user has the choice between modifying font type or color of the text and to underline it. Meta information about each mention such as the cluster id or its start index are included in the highlighting but can be hidden at the user's decision.

When the current allocation contains a huge number of mentions in general or mentions which are alone in their cluster, so called *singletons*, the excessive amount of colors can easily lead to a very unclear overview. To work around this problem there are two independent options for

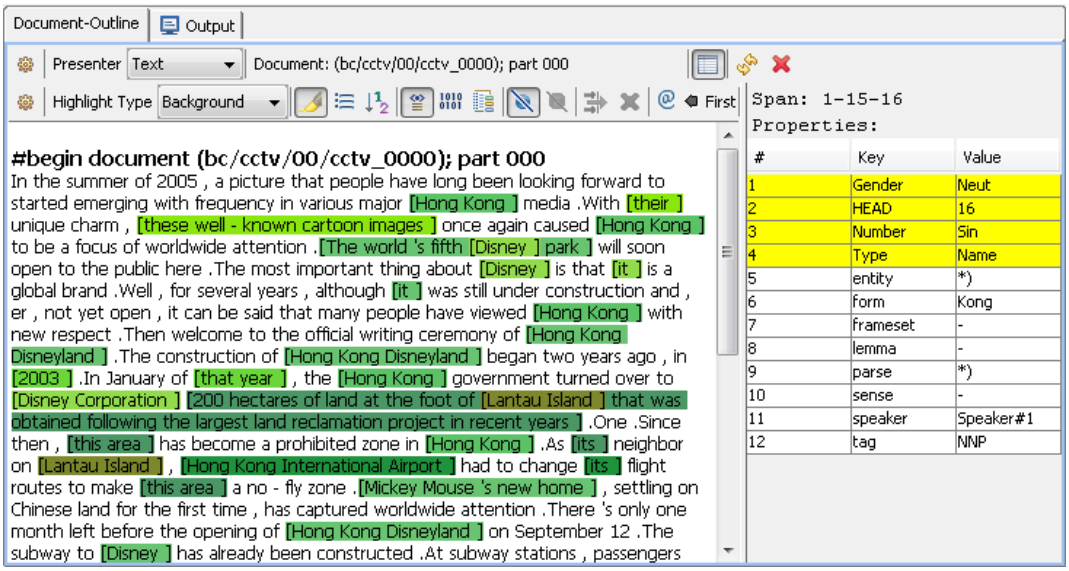


Figure 4.4: Default visualization of a single *document* in the text-based *Document-Outline*-View with mentions colorized in the text.

the user that both hide a large portion of the currently 'unneeded' highlights. First the user can choose to completely ignore all singletons in the document. Additionally, a filter option for the currently selected mention in the text is available via the upper tool-bar and the right-click context-menu in the text area.

To the right of the text outline itself is the so called *property outline* that can be (de)activated depending on necessity. It always displays information about the currently selected element of the coreference tree, in particular the set of properties assigned to that element (other visualizations show similar informations additionally via tool-tips). It is part of the general document outline and therefore available for all visualizations alike.

Graph-Outline

The second visualization option presents the coreference structure as a **tree** using a graph visualization library[2] where each cluster forms a separate sub-tree with its mentions, as shown in Figure 4.5. Each node in the graph represents exactly one mention and is labeled with that mention's cluster id, its numerical boundaries (sentence index, begin index and end index) and the excerpt of the original text it covers. Edges between cells stand for the coreference relation between their mentions. In addition to the filtering of singletons which works the same way as described for the text based visualization, there are some more highlight options available.

Since the graph view hides large sections of the documents text outside of any mention and only produces a structural point of view to the data, it might be useful for a user to see the

[2] http://www.jgraph.com/

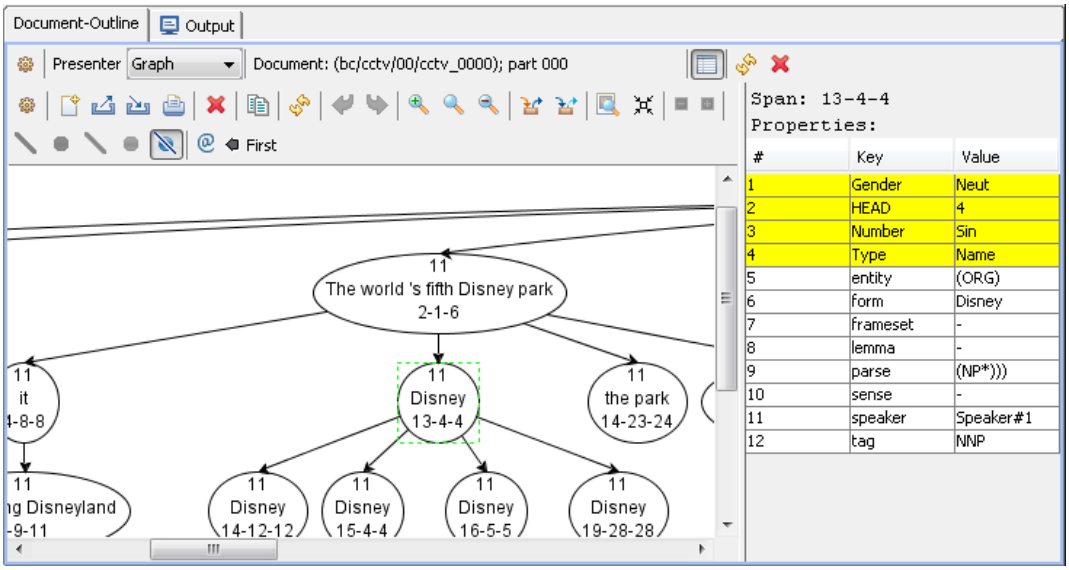


Figure 4.5: Graph-based variant of the *Document-Outline*-View with nodes representing mentions in the coreference tree.

content of a particular node in its entire textual context. A special option was added to the right-click menu of the graph component so that the user can select any number of nodes and then open a sub menu that lets him switch to the text outline with the previously selected nodes set as current filter.

Grid-Outline

The third and final visualization option is an extension of the **Entity Grid** model. An entity grid is essentially a table where each column represents an entity and each row stands for a sentence in the document. The content of a cell in this table is usually determined by the syntactic function an entity has in the given sentence (Barzilay and Lapata, 2008). Here this idea is taken one step further and the user himself is able to decide what exactly he wishes the label of a cell to represent. The default value is simply the number of times an entity is mentioned in a sentence. Via a button in the tool-bar the user can activate a text-field for text patterns to enter. These patterns will then be compiled into instructions on how to format the information of choice in each cell. There is a collection of so called *magic characters* all of which have predefined meanings. When generating the actual label for a given cell those characters will be replaced by the respective information value from the mention data. For a full list of magic characters consult Appendix A.2. An example of using a simple pattern is shown in Figure 4.7. The pattern `"b-e"` contains the two magic characters `"b"` and `"e"` which cause the *begin* and *end* index of a mention's span to be inserted.

More complex patterns are possible by using so called *property expressions* that refer to property values stored on a span or the underlying sentence data. In Figure 4.8 such an advanced pattern can be seen. The corpus in this example is in the CoNLL-2012 Shared Task

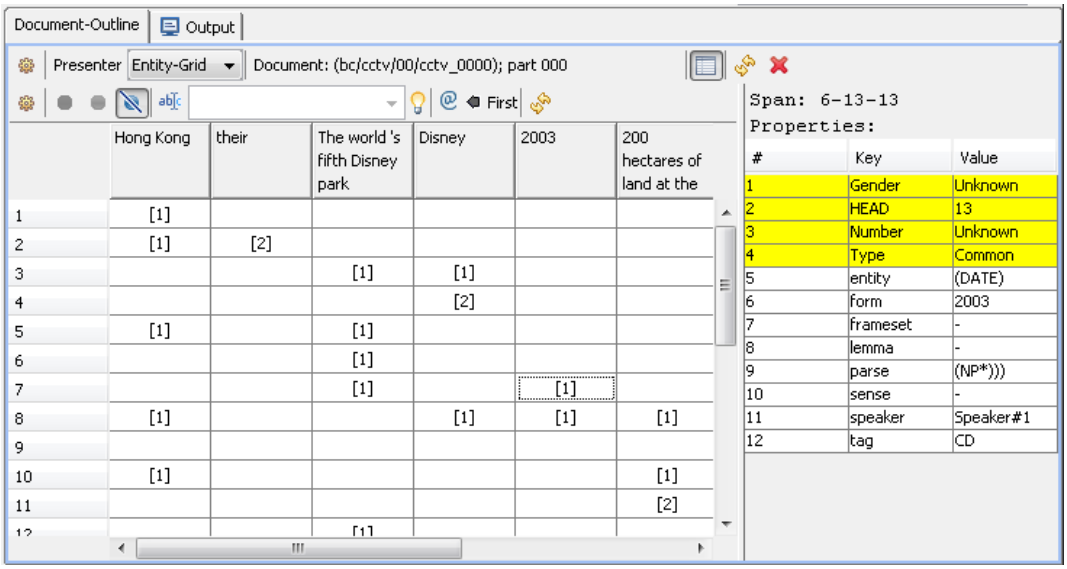


Figure 4.6: Document visualization in the style of a highly customizable Entity Grid.

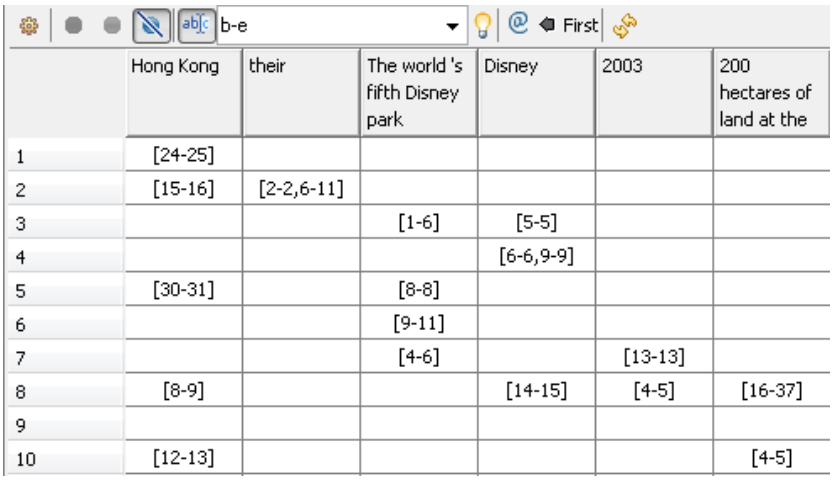
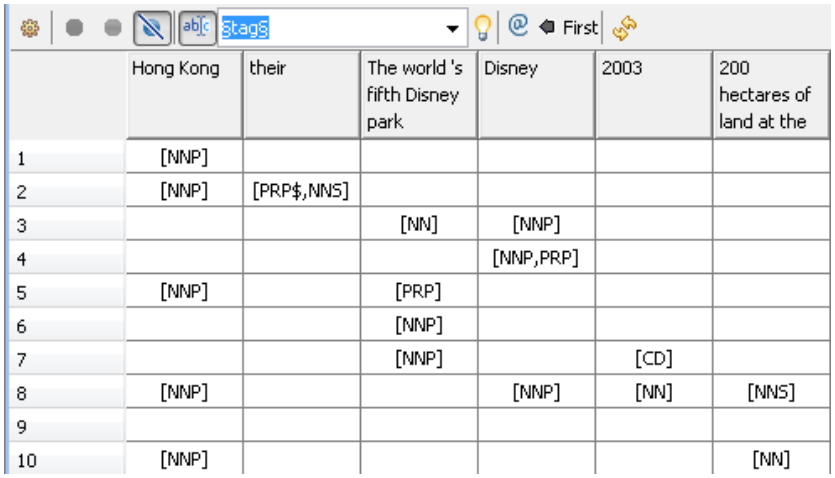


Figure 4.7: Simple example on how to use the *label pattern* mechanics of the entity grid visualization to customize the cell labels in the grid. The pattern `"b-e"` causes begin- and end-index of a mention to be used as label separated by a hyphen.



	Hong Kong	their	The world 's fifth Disney park	Disney	2003	200 hectares of land at the
1	[NNP]					
2	[NNP]	[PRP$,NNS]				
3			[NN]	[NNP]		
4				[NNP,PRP]		
5	[NNP]		[PRP]			
6			[NNP]			
7			[NNP]		[CD]	
8	[NNP]			[NNP]	[NN]	[NNS]
9						
10	[NNP]					[NN]

Figure 4.8: A more advanced way of using a pattern to customize cell labels. In this case the underlying sentence will be queried for the property *"tag"* at the index that is defined as a mention's head.

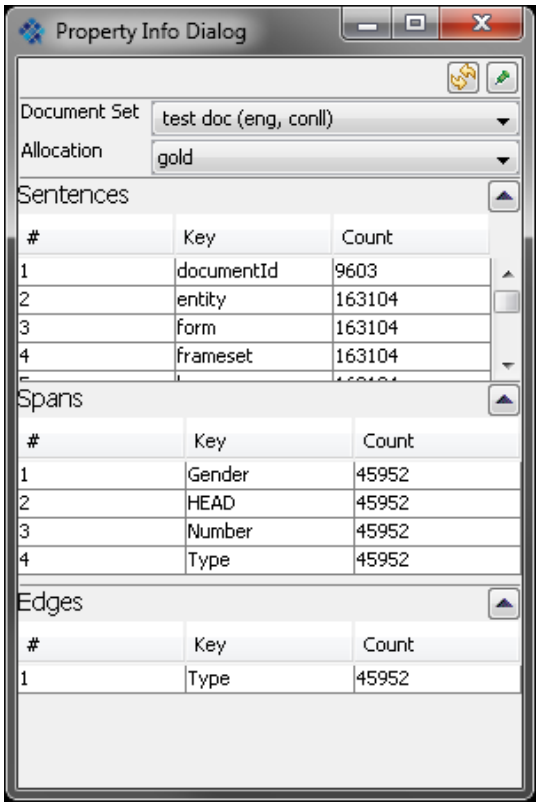


Figure 4.9: A useful utility tool to collect and display available property keys for a given combination of *document set* and *allocation*.

format and the `"§tag§"` expression used in the pattern links to the value of the *tag* property of the word that is marked as the head of the span. Similarly to the graph-based outline, the grid allows the user to easily switch back to the bare text outline with the spans of a cell filtered.

To use the pattern engine to its fullest extent, the user needs knowledge of the exact property names used in both the allocations and the corpus itself. Since the data model (see Section 5.2) poses no restrictions on the number and naming of properties, the user would have to manually search the data files in question for occurrences of interesting properties to acquire the exact name or to learn about possible properties in the first place. A special utility tool accessible via the coreference manager's tool-bar was developed to avoid this additional effort on the user's side. In a special window shown in Figure 4.9 the user can, in the same way as in the explorer view, combine a document set and an allocation. The tool then traverses both the raw corpus data and the allocation and collects all the property names that are used. After completion it presents the user with a list of these property names split by their target type (word in a sentence, span or edge) combined with an occurrence count. Double-clicking a row in one of those lists directly copies the corresponding property name in the system clipboard so that users do not have to transcribe them for use in the label pattern text-field.

4.2 Querying and Exploration

While the features described in the previous section allow the user to comfortably inspect corpora and their coreference structures, they are by no means sufficient when it comes to more complex exploration needs. The ICARUS platform ships with a dedicated plugin that manages search operations and, in that regard, provides a wide range of graphical interfaces to the user. Figure 4.10 shows the main perspective of this *Search-Tools* plugin. The area to the left is dedicated to managing a search by selecting the type (which in the current case will be a search on coreference structures that returns documents as result items) and target of it. Valid targets are all the corpora previously registered via the coreference manager.

When the basic choices regarding type and target are made, the user can continue to express his exact query using the *Query Editor* view on the right. A screenshot of this editor is shown in Figure 4.11. The editor is split in two components, the upper one holds the graphical representation of the query and the lower one displays the query in textual form. Which way of defining a query to use is up to the user. Graphical and textual queries can also be converted into one another by means of a single mouse-click. The query shown in the screenshot is intended to investigate possible values for the *"Type"* property of nodes that refer to nodes tagged as *"Name"*. Note the usage of the special grouping operator <*> in the constraint of the second node that refers to the property *"Type"* on the node level. This operator collects the instances of a given constraint in the target tree when matching the query tree and groups them. Search results created that way can then be viewed as frequency lists or tables depending on the number of grouping operators in the query. Specialized visualizations are available for up to three groups and include lists and tables to properly outline the frequency breakdowns.

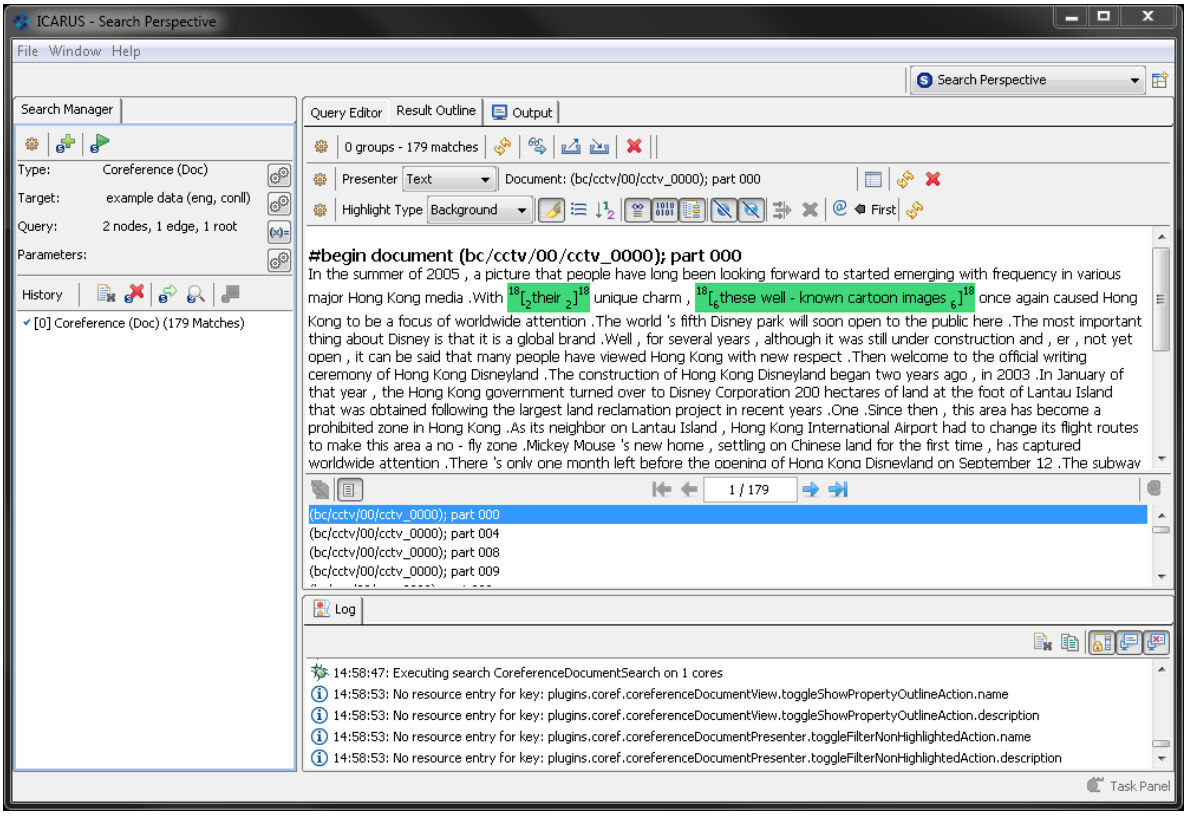


Figure 4.10: Complete window currently showing the *Search-Perspective* where a user can define, execute and manager search operations and view their results.

After executing the search, the user can then browse the search results in the *"Result Outline"*. Figure 4.12 shows the result breakdown of the query mentioned earlier in this section. Since there was exactly one grouping operator involved the result will be presented by means of a frequency list containing all the instances that have been found using the constraint the grouping operator was used for. Next to each instance a cell lists the exact frequency of its occurrence. By double-clicking this cell the user selects a portion of the result that will then be presented to the right. Using the exact same visualizations as the document outline in Section 4.1.3, the result outline preserves the same flexibility in terms of how particular items in the result will be presented. In addition to the basic features of the document outline, the user can filter out all coreference members (nodes or edges) in the visualization that are no immediate part of the result structure (i.e. they do not map to definitions in the query).

4.3 Error Analysis

The Sections 4.1 and 4.2 have shown how document sets with a single allocation can be browsed and explored both manually and with the help of the search engine. To address specifically the needs of NLP tool developers and other users who are interested in a comparison of two

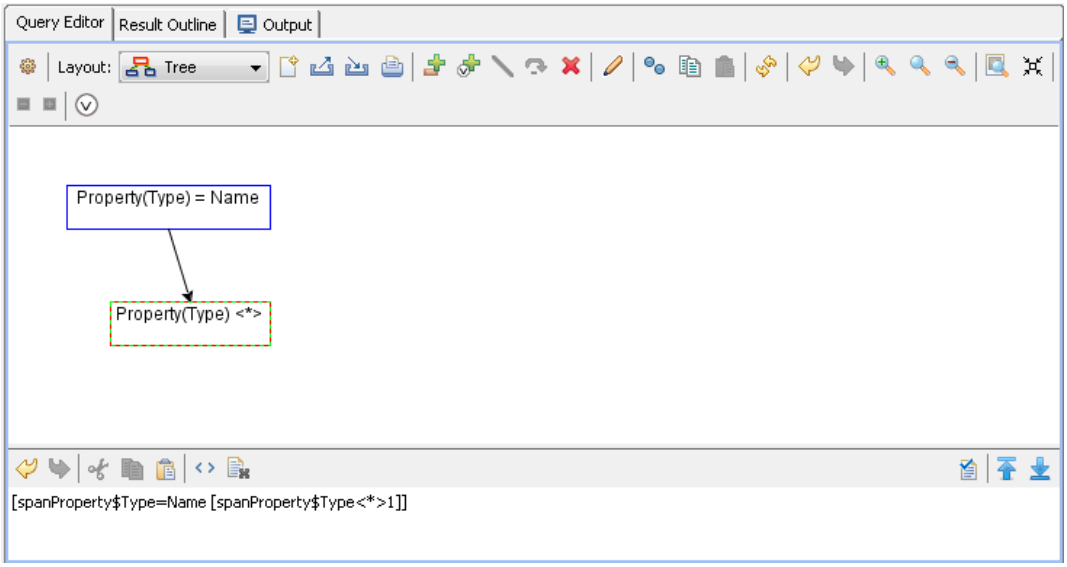


Figure 4.11: Editor for defining queries both in a graphical and text based way. The query in the screenshot is used to generate a quantitative summary for the *"Type"* property of all the immediate children of nodes whose *"Type"* is set to *"Name"*.

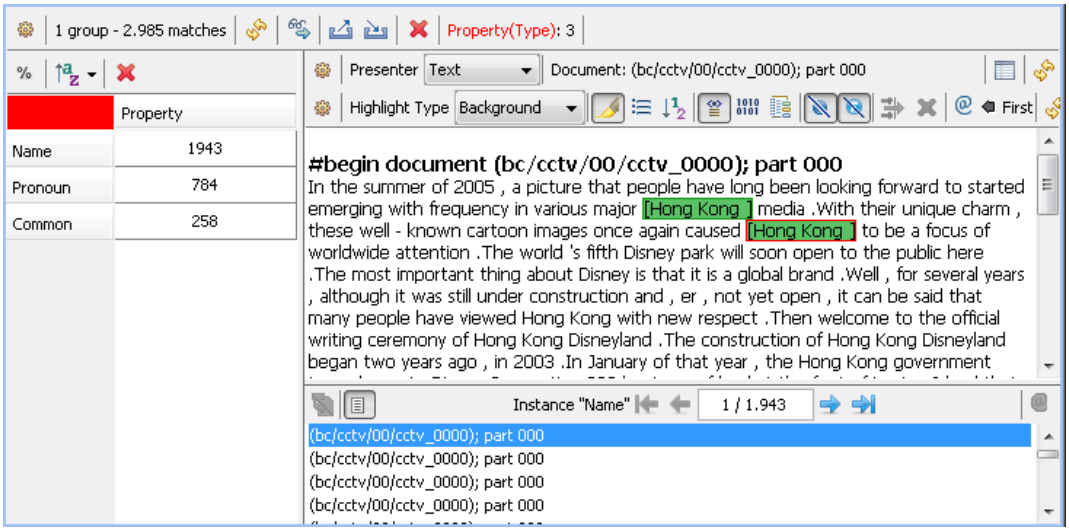


Figure 4.12: Result of a so called 1-dimensional query, motivated by the number of grouping operators <*> used in it.

different allocations, additional highlighting features have been implemented for two of the visualization types introduced above. When the user assigns a second allocation to a document set via the explorer, both the graph- and grid-outline offer a special highlighting mode. A set of buttons on their respective tool-bars (de)activates the graphical emphasis of nodes and/or edges (the latter only in the graph-outline) that are found to be false positives or false negatives.

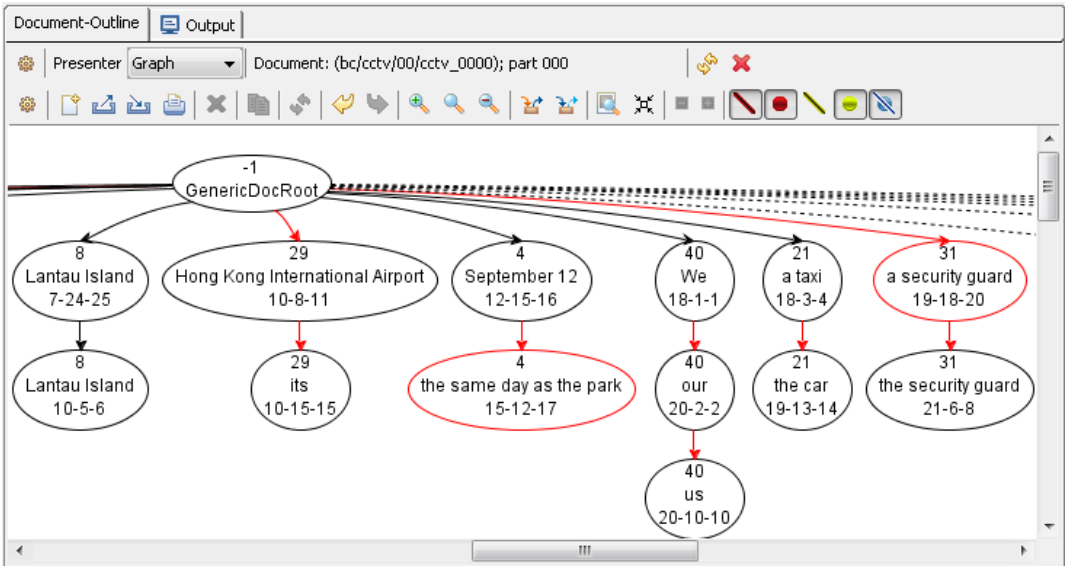


Figure 4.13: Example of the highlighting features of the graph-outline supporting manual error analysis. False positives are marked as red and false negatives appear with a dashed stroke-style.

In the graph-outline, the decision can be made for nodes and edges independently since crossing edges that indicate false negatives can easily lead to confusion. As is obvious from Figure 4.13 false positives are marked in a typical red and false negatives get a dashed stroke-style. Similarly to this, but without the visualization of edges, the grid-outline in Figure 4.14 lets the user decide whether or not to mark false positives with a red label and false negatives with a green one.

To perform a quick check for errors or mismatches of some automatically created coreference data the user only has to assign the corresponding allocations to a document set and then chose amongst the two outline types. By simply scrolling through the outline, the user can quickly identify errors and investigate them further.

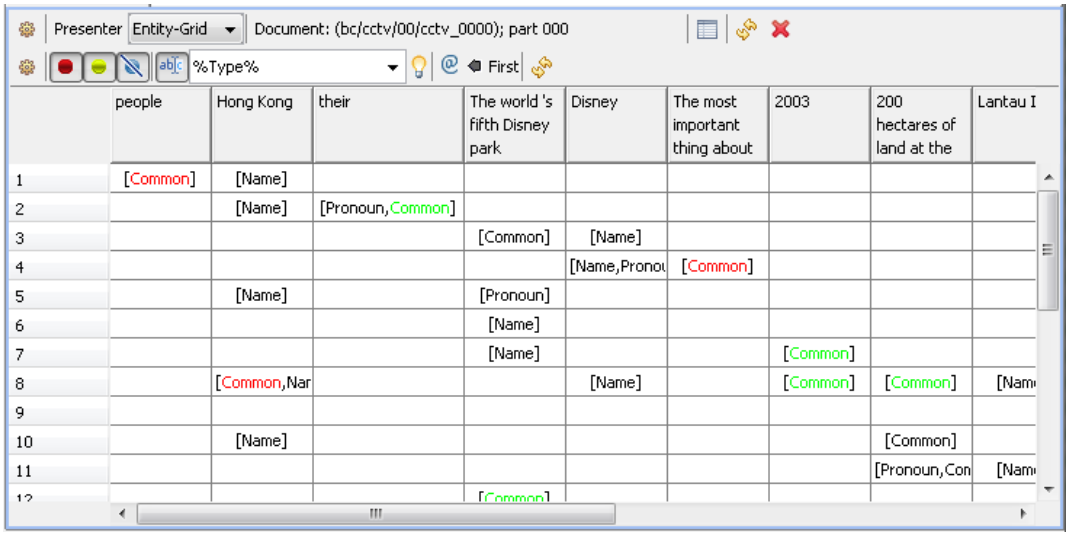


Figure 4.14: Manual error analysis using the grid-outline's *compare* feature. Spans found to be false positives are marked in red and false negatives are colored green.

5 Architecture

IEC is essentially a plugin for the open-source tool platform ICARUS(Gärtner et al., 2013). This chapter is intended to provide an overview of this platform and the components relevant to the thesis. Section 5.1 lists the core features and structure of ICARUS and in Section 5.2 I will outline the main aspects of the data model used by IEC.

5.1 Features

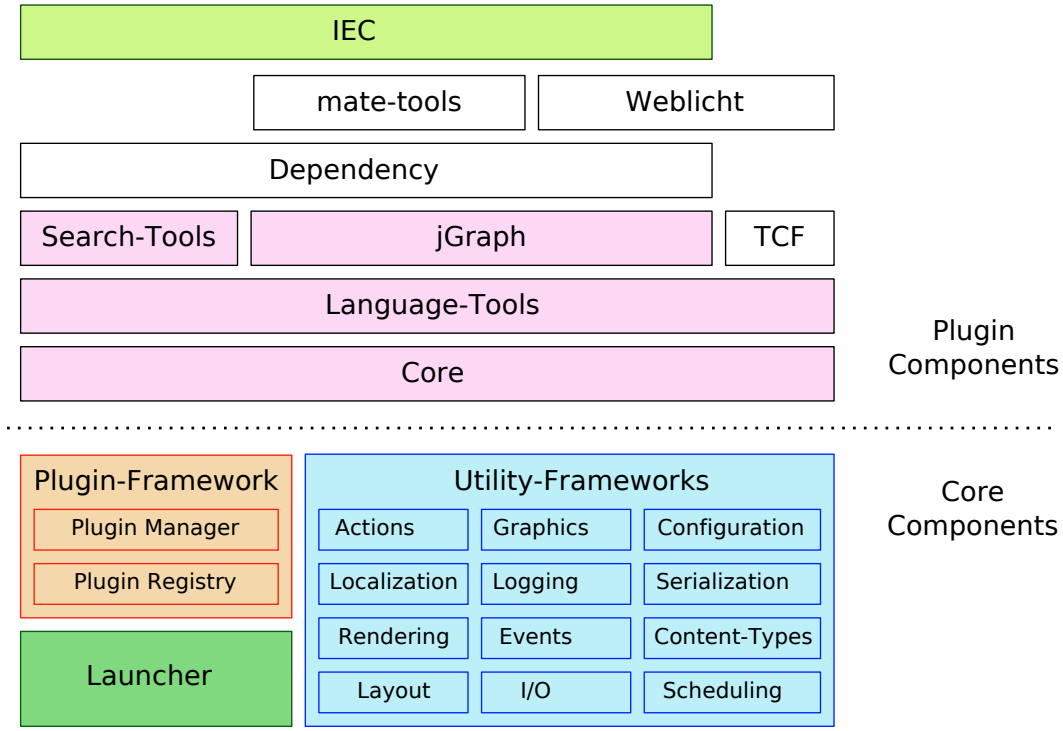


Figure 5.1: Components of the ICARUS platform with members of the core framework in the lower area and plugins contained in the current distribution above. Colored plugins are prerequisites of the IEC plugin.

The ICARUS platform is a collection of frameworks and tools with a plugin-based architecture. It is focused primarily on interactive visualization of data used in NLP tools and the ability to explore said data with its customizable search engine. Figure 5.1 provides a quick overview

of the main components that form the ICARUS platform and their hierarchic relations. The collection of core components in the lower part of the figure consists of three modules of which only two will be covered here since the launcher module's responsibility does not transcend the platform start-up:

Plugin-Framework Centered around the open-source Java-Plugin-Framework (JPF)[1] this module manages the registration, validation and dependency checks of all plugins. In addition, it monitors the life-cycle for each active plugin and performs lazy activation, that is it only activates a plugin as soon as its content is actually required. Plugins are registered using a manifest file in XML format. This manifest describes all elements and properties of a single plugin. A very important part is played by the *extension-points* and *extensions* a plugin declares in its manifest. Extension-points describe well-defined adapter points which other plugins can connect to by an extension declaration. This declaration passes the required arguments on to the extension-point.

Utility-Frameworks The vast majority of code in the platform core outside the aforementioned plugin-framework is organized in a multitude of frameworks that greatly simplify the process of developing and managing user interfaces.

The upper part shows the plugins that ship with the current distribution of the platform. Their arrangement reflects a prerequisite-hierarchy where a plugin depends on all plugins located directly below the rectangle holding its name. IEC on top is an exception of this and depends only on the other colored plugins. These prerequisites of IEC will be outlined in the following:

Core The core plugin manages the basic visual appearance of the platform. It organizes windows and defines the frameworks and mechanics used for communication between different plugin components within the same window. Additionally, it serves as a kind of bridging point between some of the core frameworks of the platform and new plugins to ensure proper initialization of the first and correct usage by the latter. Two of the most important and very frequently used extension-points that the core plugin defines are `View` and `Perspective` which are vital in the structural organization of the user interface. Both of them are used by the IEC plugin, too. While a `View` serves as container for a single specialized interface component or tool, like for example a list with log entries or a graph visualization of some data, it is always a part of an enclosing `Perspective` which in turn arranges multiple `View` objects associated with a certain context.

Language-Tools In principle a large collection of constants and very general interfaces intended to allow other plugins to define their own data models and to keep them compatible with other tools.

Search-Tools Being the by far largest plugin of the ICARUS platform, it hosts all the fundamental interfaces and tools related to search operations. Although the basic data models all refer to general (graph) search applications, most of the default implementations are restricted to tree structures. The plugin provides a multitude of components for user

[1] `http://jpf.sourceforge.net/index.html`

interfaces that can be used to manage multiple searches, define and edit search queries both graphically and textually, and to visualize search results depending on the number of grouping operators in the query. Example screenshots of these components together with some content were presented in Section 4.2.

jGraph Centered around the identically named open-source graph visualization library jGraph[2], this plugin focuses on integrating graph visualization into the platform's user interface. It defines several components that can be customized by other plugins to meet their requirements in presenting different types of data. It also provides other plugins a large number of predefined utility functions that greatly increase their usability. Being able to manually adjust an automatic graph output or to export parts of a graph as a graphic or into XML are only a few examples.

5.2 Data Model

The IEC plugin is composed of many data types that model various aspects of coreference structures and the raw corpus data underneath it in a hierarchical fashion. In the following sections I will explain how the two main components of that hierarchy are designed and how they preserve the flexibility that is needed to model the wide range of coreference formats available.

Document

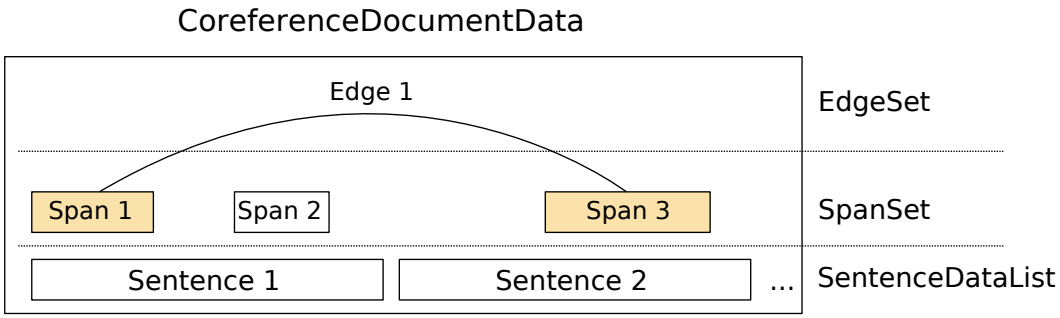


Figure 5.2: Hierarchical data model of a single coreference document.

IEC uses a collection of data structures ordered hierarchically to represent the various elements involved in a coreference model. The sentence model was designed as flexible as possible in order to not pose restrictions on the levels of annotation a certain *base data* corpus may already contain. The `CoreferenceData` is an extension of the original sentence format that the ICARUS platform provides, which only served as a logical grouping container for the word tokens in a sentence. The derived implementation expands the container by the ability to

[2] http://www.jgraph.com/

assign an unlimited and unrestricted amount of annotations in the form of *key-value* pairs to each word. The actual coreference structure is built on top of this sentence model. Figure 5.2 shows the types and levels involved in the representation of a single document. The base data is organized in a list of sentences. Above those all the mentions are represented as `Span` objects consisting of information about the span of text they cover and a reference to their assigned `Cluster`. As mentioned in Chapter 1, there are many different sets of attributes the various groups involved in design and annotation of coreference data assign to both nodes and edges in a coreference graph. Therefore the `Span` type supports *key-value* attributes to model arbitrary information very much like the abstract sentence representation.

The collection of edges that represent coreference relations are placed on top of the `Spans`, each holding references to their source and target `Span` and providing the same support for additional property data. Between the `Span` and `Edge` level exists the `Cluster` model as a sort of utility type. It holds all the `Span` objects belonging to the same cluster (referring to the same entity) and for each such `Span` the incoming `Edge` in the coreference tree if the node is not the root of a sub-tree. To allow for easy traversal and to provide a logical organization, the elements of each level are grouped in list-like structures as can be seen on the right of the figure. Both the data models on the bottom and the top of this hierarchy are introduced as `interfaces` with various predefined possible default implementations to enable future extensions to integrate additional capabilities without the need to redefine the entire data model. Note that the default implementations do not store the `SpanSet` and `EdgeSet` objects themselves but query their surrounding *document set* container (see below) when required.

Document-Set

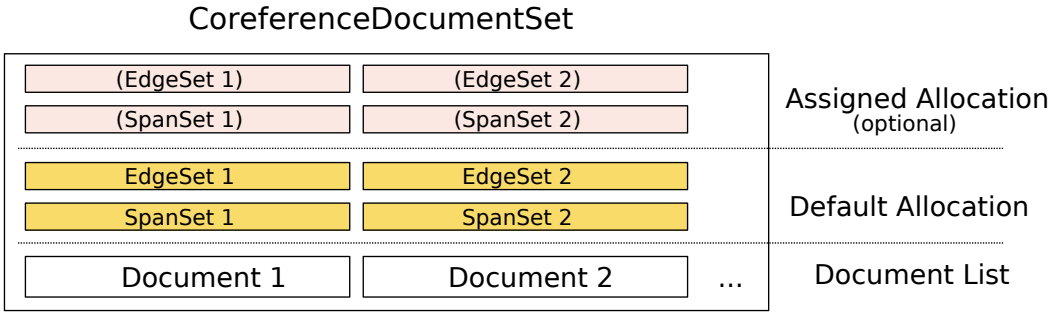


Figure 5.3: Hierarchical data model of a set of coreference documents and the possible allocations.

The next step into the bigger hierarchy is the grouping of several documents into a single `CoreferenceDocumentSet` that forms the actual corpus. Its main purpose is to provide an ordered view on the document objects and to manage the so called *allocations* that can be assigned to a single document set. Each `CoreferenceDocumentSet` can contain up to two allocations. The *default allocation* is created when the underlying corpus is loaded and cannot be modified afterwards. This allocation will only be available when the format of the corpus already supports coreference information as for example the one specified by the CoNLL-2012

Shared Task.[3] If this is not the case, the default allocation will stay empty. The second allocation enables other parts of the software to programmatically change the allocation that should be used for the document set. This is however not to be confused with the allocations the user can manually assign when using the various tools of IEC.

[3] http://conll.cemantix.org/2012/data.html

6 Evaluation

In this chapter I present discussions on how IEC meets the various requirements that different users impose on a utility tool for interactive visualization of coreference annotations. The two most notable user groups will be represented by corpus linguists and NLP developers, respectively. Data being used in this chapter is obtained from the CoNLL-2012 Shared Task development set (Pradhan et al., 2012) and annotations are created using an IMS-local implementation of the system of (Fernandes et al., 2012).[1] The *gold* coreference trees were produced by the same system, but applying it in a constrained setting which only outputs correct trees.

6.1 Phenomena Exploration

For **corpus linguists** or various non (computational) linguists from other disciplines, the focus of interest lies in intuitive visualizations that allow a comfortable inspection of the data without special technical knowledge being required to use the tool in question. In addition, the user should be provided with different visualizations where possible and given the ability to comfortably query a corpus for instances of a specific phenomenon. Depending on the task, it might be necessary to switch back and forth between these different views on the data. So far, IEC is perfectly suited to satisfy the needs of anybody intending to browse through coreference annotated data. It is easy to use, requires no installation and is by design of the ICARUS platform extensible in a lot of ways. Therefore anybody can integrate new readers for other corpus or allocation formats.

Besides providing the user with informative visualizations, it is vital to aid him in the search for examples of certain phenomena that he is interested in. The search engine IEC ships with allows the user to define queries both in plain text and graphically, the latter being much like *"drawing"* a tree representation of the phenomenon. Constraints range from properties and structural relations on nodes or edges in the tree, down to word tokens in the underlying text and all the additional annotations in the corpus that are encoded via properties on the sentence level. The search supports a large collection of operators and can handle negations and disjunction in the query expressions.

A very straightforward application in the context of corpus linguistics is the search for certain rare phenomena in a large corpus that would require immense human effort to search manually. *Forward references* or *"cataphora"* have been described in Chapter 1 and are a usually rare

[1] Thanks to Anders Björkelund for providing this.

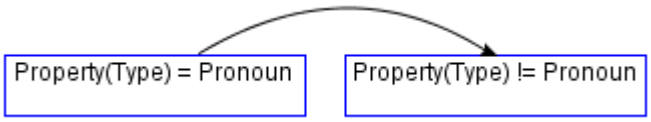


Figure 6.1: A very simple query to look for cataphora constructions involving a pronoun referring to a non-pronoun.

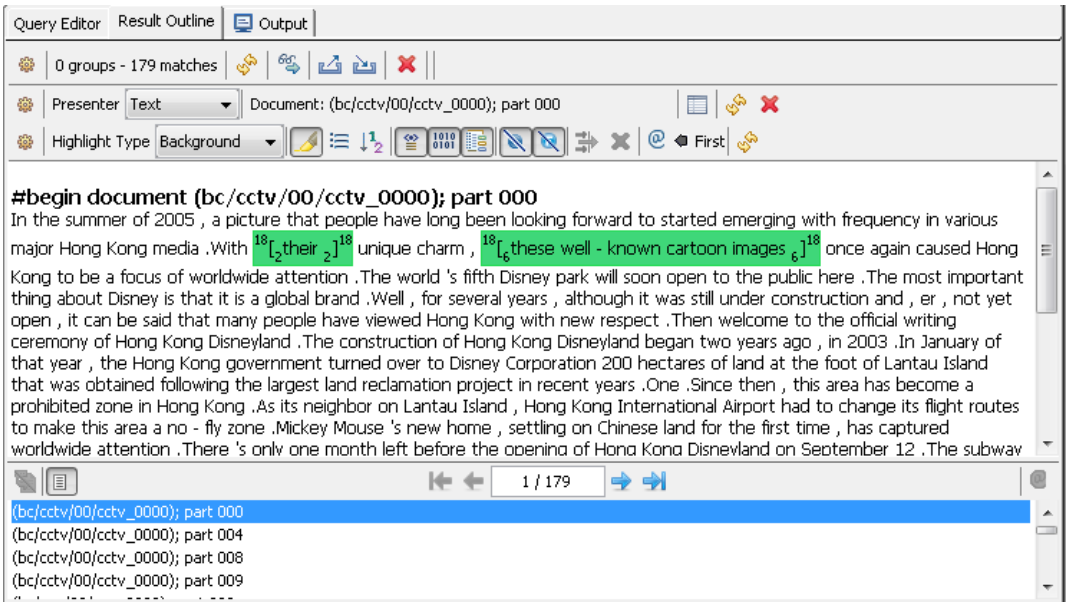


Figure 6.2: Result of the query in Figure 6.1 applied to an example corpus. The list shows documents containing cataphora.

construct. The following example outlines a possible approach, assuming the user is familiar with the nature of the phenomenon but has no technical knowledge about the particular corpus and allocation formats being used. Cataphora typically involve a pronoun referencing a later introduced non-pronoun. With help from the explorer and the property outline from Chapter 4 the user figures out the property name being used in the data is `"Type"`. The constructed query is shown in Figure 6.1 and executing the search yields the result outline in Figure 6.2.

Besides finding instances of a particularly rare phenomenon, the search engine can be used to determine the "rareness" of a given construct. *"How often does phenomenon X occur in my corpus? in a single document? among all mentions or clusters?"* are typical questions asked by corpus linguists when investigating a corpus. The ability to generate quantitative breakdowns of search results aids in answering those questions. Using the above example of finding cataphora, a collection of further queries could be defined to gather additional data to form an actual statistic. Typical examples for such additional information would be the total number of pronouns and clusters to see how frequent cataphora are in different contexts.

6.2 Error Detection / Property Exploration

The needs of **NLP developers** and analysts as the second group are often more complex and require a more detailed discussion. Naturally, the demands of proper visualization and convenient usability as stated above persist. A common issue regarding development of NLP tools is the investigation of errors or false predictions. Utility tools can aid in this matter by providing either a dedicated exploration capability that allows (semi) automatic analysis of the data or by means of some comparison being created between predicted data produced by the NLP tool and a given correct version. This section is focused on using the search engine for automatic detection of systematic errors in the output of a coreference resolver.

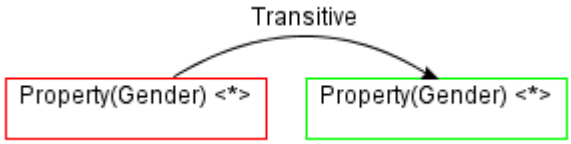


Figure 6.3: So called 2-dimensional query, motivated by the number of grouping operators <*> used in it. This query is used to generate a quantitative summary for all the existing *gender* properties of nodes that are either start or end of a path in a given coreference tree.

To show how the query functionality in IEC can be used to facilitate complex analysis and error detection, I will use the issue of **gender-mismatches**. It should be noted that the gender assignments used in this chapter were assigned automatically and that sometimes the automatic assignments are wrong. Ideally, all the mentions referring to the same entity should also be of the same gender, or at least form a compatible combination. The term *"compatible"* in this context is motivated by the fact that there are legal combinations of mentions sharing the same cluster without holding the exact same gender tag. Reasons are that it is not always possible for an automatic system to determine the gender of a mention (therefore assigned `"Unknown"` in the allocation used for this section) or that an entity introduced as generic `"Neutral"` is later referenced by an actual `"Female"` or `"Masculine"` mention. Real mismatches are clearly represented by mixed occurrences of masculine and feminine references in a single cluster.

	Neut	Unknown	Masc	Fem
Neut	5978	463	143	30
Unknown	289	17899	339	239
Masc	152	1623	8836	162
Fem	21	301	69	1371

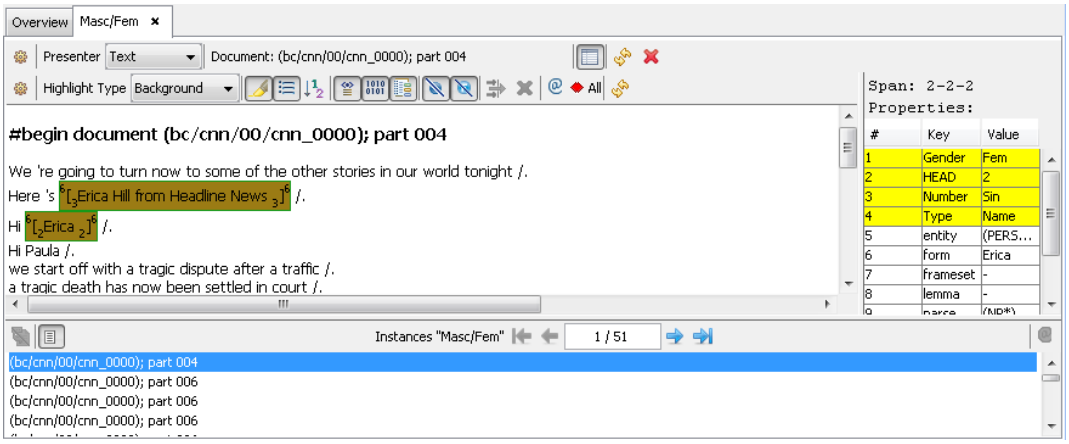
Figure 6.4: High-level result overview of the query in Figure 6.3. Numbers in a cell indicate how often an instance of the search query was found in the target with grouping constraints replaced by the actual values of the given row and column labels.

A very simple query that makes use of the value distribution feature of the search engine in IEC is shown in Figure 6.3. It will match any two nodes in the coreference tree that are connected by a path (the transitive edge represents a path of arbitrary length).

	Neut	Unknown	Masc	Fem
Neut	3364	202	62	5
Unknown	146	4511	72	16
Masc	78	226	2865	51
Fem	8	58	18	449

Figure 6.5: A result very similar to the one shown in Figure 6.5 but restricted to directly connected nodes. This outline can be used to quickly check for errors in a coreference resolver by looking at the cells that signal a relevant mismatch (like *"Masc/Fem"*).

The tabular outline in Figure 6.4 presents the result of the gender-query on an automatically created output from the resolver. The value in a cell states the number of times there was a path from a node with the gender property of the row label to a node with a gender property as given by the column label. As one would expect the majority of combinations is concentrated on the diagonal axis but some noisy gender assignment is expressed by the table. Of special interest are the cells that signal an extreme gender-mismatch like an entity that is introduced as being female and referred to by a node that is tagged as male or vice versa. After knowing



#	Key	Value
1	Gender	Fem
2	HEAD	2
3	Number	Sin
4	Type	Name
5	entity	(PERS...
6	form	Erica
7	frameset	-
8	lemma	-

Figure 6.6: *Detail Outline* of the frequency table in Figure 6.5 listing all the actual instances in the corpus that contributed to the count for the selected gender combination *"Masc/Fem"*.

the general distribution of gender relations in clusters, the user might be interested in the gender properties of two directly connected nodes. By revoking the transitive feature from the single edge in the above query while leaving the rest intact the search yields the distribution of Figure 6.5.

The outline in Figure 6.6 shows examples in the investigated allocation for *female* nodes referencing *male* nodes. When looking at the example, it is obvious that the assignment `"Masc"` on the phrase *"Erica Hill from Headline News"* is erroneous, thus causing a gender-mismatch with respect to the correctly interpreted name *"Erica"* later in the sentence (which is correctly labeled female).

6.3 Error Analysis / Annotation Comparison

The previous section described the detection of fundamental mismatches regarding gender by using the search engine of IEC. I will now introduce another important use case of the framework for NLP developers, that is **error analysis** by means of comparison between gold and predicted annotations. In order to understand the nature of a given mistake made by a system, it is often essential to have a look at structural differences between the erroneous output and a gold version. Hereinafter the various tools of IEC which can aid in that matter will be outlined. As example I take a very short document containing several mistakes made by the resolver.

Predicted Annotation

Two months after [a bomb]$_{a_1}$ blasted [[its]$_{a_2}$ hull]$_{b_1}$, [the "USS Cole"]$_{c_1}$ is back in the United States. [The damaged ship]$_{c_2}$ was carried from Yemen aboard a Norwegian transport ship. 17 sailors were killed when [the ship]$_{c_3}$ was attacked by suicide bombers in the port of Aden. [A temporary patch]$_{d_1}$ has already been made to cover [the 40]$_{e_1}$ by 40 hole in [[the ship's]$_{c_4}$ hull]$_{b_2}$. [It]$_{d_2}$ will be welded to [the ship]$_{c_5}$ before [it]$_{c_6}$ is unloaded from [the carrier]$_{e_2}$. Repairs in Mississippi are expected to cost more than $ 150 million and last a year at a ship-building facility in Pascagoula, Mississippi.

Gold Annotation

Two months after a bomb blasted [its]$_{f_1}$ hull, [the "USS Cole"]$_{f_2}$ is back in the United States. [The damaged ship]$_{f_3}$ was carried from Yemen aboard [a Norwegian transport ship]$_{g_1}$. 17 sailors were killed when [the ship]$_{f_4}$ was attacked by suicide bombers in the port of Aden. [A temporary patch]$_{h_1}$ has already been made to cover the 40 by 40 hole in [the ship's]$_{f_5}$ hull. [It]$_{h_2}$ will be welded to [the ship]$_{f_6}$ before [it]$_{f_7}$ is unloaded from [the carrier]$_{g_2}$. Repairs in Mississippi are expected to cost more than $ 150 million and last a year at a ship-building facility in Pascagoula, Mississippi.

Figure 6.7: Annotations for an example text. The upper highlighting shows the result of an automatic resolution, while the lower version represents a gold annotation. To improve readability all singletons in the annotations are left unhighlighted.

Figure 6.7 shows the textual visualization of the two annotations in question. To not confuse the reader, I left out all singletons in both annotations so that primarily relevant mentions are highlighted. Where applicable the same color is used for clusters in the different annotations that actually refer to the same entity and are not part of a general mistake made by the resolver. The visual differences in the two highlighted texts clearly hint on some serious mistakes made by the resolver, but a more structural inspection and comparison is necessary to pinpoint the exact types of errors.

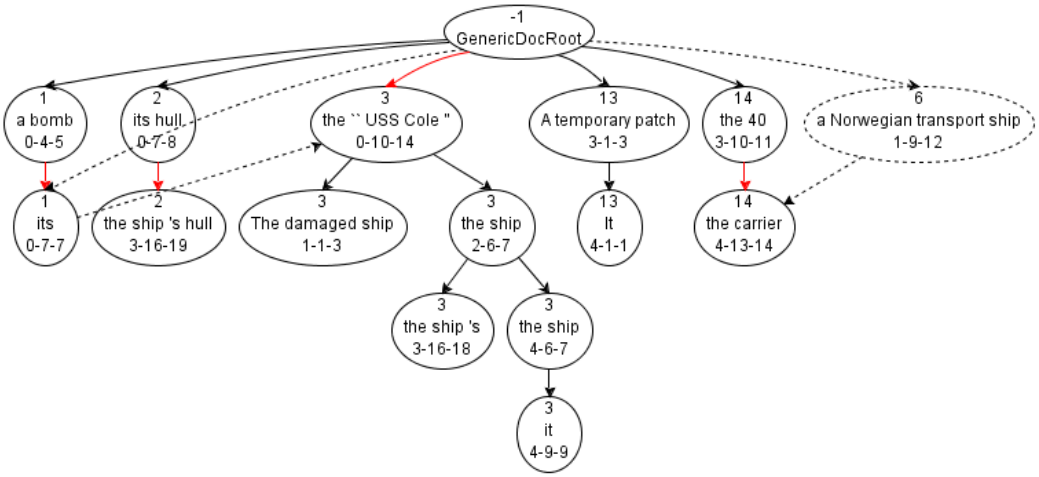


Figure 6.8: Graph outline showing the two annotations from Figure 6.7 in a comparative view. Red edges mark false positives in the predicted output and dashed edges or nodes signal false negatives. Again singletons of both the gold and predicted annotations are hidden to not obscure the visualization.

To achieve the required visualization, the user can assign both annotations as allocations to the document set in question and activate the comparison features of the graph-outline. The result of this is illustrated in Figure 6.8. Solid lines mark nodes and edges of the predicted allocation. The content of each node is the cluster id as assigned by the resolver, the part of text covered by the mention and a numerical representation of the mention itself. Furthermore, there are special treatments for false positives and false negatives in the prediction. Edges colored in red signal a false positive mistake of the system, and parts of the graph rendered with dashed lines are false negatives that the resolver missed. While the error outline might already prove useful for the developer, a deeper understanding of the reasons for a particular error is required to fix it. Looking at the graph-outline there are several issues in the predicted annotation.

The first relates to the clusters $\{$a bomb, its$\}_{(a_1-a_2)}$ and $\{$the "USS Cole", the ship, the ship's, $\ldots$, it$\}_{(c_1-c_6)}$. With help from the textual visualization and the properties outline, it becomes obvious that the mistake is caused by a cataphoric relation. At the point where the resolver has to assign the referent of the mention *"its"*$_{a_2}$, the only possible target is *"a bomb"*$_{a_1}$. This is because the mention-pair model only considers the mentions located before the current item. This type of mistake can be quickly recognized in the graph when there is a dashed path (visiting a pronoun) that is an alternative for a single red edge.

Another mistake presented by the comparison view is the faulty assignment of the mention *"the carrier"*$_{e_2}$ to the cluster of *"the 40"*$_{e_1}$ instead of *"a Norwegian transport ship"*$_{g_1}$. Since the resolver has no semantic knowledge it has to rely on other features. While it recognizes the entity of the transport ship by means of a mention (this can be checked by deactivating the filter for singletons), it is not able to form a relation when considering the mention *"the carrier"*.

	a bomb	its hull	the `` USS Cole ''	a Norwegian transport ...	A temporary patch	the 40
1	[Common,Pronoun]	[Common]	[Common]			
2			[Common]	[Common]		
3			[Common]			
4		[Common]	[Common]		[Common]	[Common]
5			[Common,Pronoun]		[Pronoun]	[Common]
6						

Figure 6.9: Grid-Outline of the two annotations in Figure 6.7 in comparison mode.

For reasons of completeness, Figure 6.9 shows the same comparison using the grid-outline. Compared to the graph-based visualization, a major drawback is the lack of structural information expressed by edges. It does however offer a much more compact view on the annotation. In addition, the ability to change the actual labels used for cells in the table grant a high degree of flexibility. This way the user can perform his analysis in a much more target-oriented manner. This allows for easy manual detection of errors when browsing a document. It does however still require human effort for searching, since the search engine is unable to handle more than one annotation simultaneously. To enable such a feature a considerable amount of redesign on the ICARUS platform level is required, and I left this for further work.

7 Conclusion

I have presented IEC, an interactive tool for exploration and analysis of coreference annotations that is realized as a plugin for the tool-platform ICARUS. It is aimed at a wide user group comprising not only (computational) linguists in the field of corpus linguistics or natural language processing. Depending on their individual focus of interest, these users impose different requirements on the tools they work with. Such requirements include the way and complexity of visualization and exploration capabilities and have been outlined in detail in Chapter 2. IEC has been specifically designed to satisfy the combined needs of both (corpus) linguists and (NLP) developers as representatives of the two main groups of potential users.

IEC enables users to visualize coreference structures in one of tree ways:

1. a plain text-outline with graphical highlighting of mentions
2. a graph-based diagram modeling a coreference tree
3. an advanced implementation of a customizable entity grid

The different visualizations offer fast browsing of coreference annotations. The ability to switch between these views on a given set of data avoids locking the user into a single way of visualization. Both the data model and modules used for visualization are designed in such a way as not to pose restrictions on the data being processed.

In order to outline how IEC can be used to satisfy the needs of different users I have presented several use cases. The scopes of application have been interactive exploration of corpora and the detection and analysis of errors using various tools provided by the framework. For all three of these applications IEC has proven to be a proper solution. This applies especially to the ability to present both annotations and search results in several different views, respectively.

The tool supports the corpus format used in the CoNLL-2012 Shared Task (Pradhan et al., 2012) and an additional proprietary allocation format to define mentions and their links. In conjunction with the extensible platform design of ICARUS it is however straightforward to integrate additional formats with very little effort.

A Appendix

The following two sections provide a more technical description of formats introduced by IEC. Appendix A.1 outlines the default format used to read allocations (the default format for document sets is the same as defined in the CoNLL-2012 Shared Task[1]). In Appendix A.2 I give an overview on the syntax of *label patterns* which can be utilized to customize the grid-outline introduced in Section 4.1.

A.1 Allocation Format

The default format for reading coreference allocations is a very compact list structure as shown in the following text which contains a shortened example of the nodes and edges for a single document in the referenced corpus:

```
#begin document (bc/cctv/00/cctv_0000); part 000
#begin nodes
ROOT
0-2-5   Gender:Neut;HEAD:3;Number:Sin;Type:Common;
0-5-5   Gender:Unknown;HEAD:5;Number:Unknown;Type:Common;
...
26-26-26   Gender:Neut;HEAD:26;Number:Sin;Type:Name;
#end nodes
#begin edges
ROOT>>0-2-5 Type:IDENT
ROOT>>0-5-5 Type:IDENT
...
7-14-15>>26-2-3 Type:IDENT
9-12-13>>26-5-6 Type:IDENT
...
ROOT>>26-26-26 Type:IDENT
#end edges
#end document
```

The opening line contains the unique id `"(bc/cctv/00/cctv_0000); part 000"` of the document as defined in the corpus this allocation is referring to. The section of text delimited

[1] `http://conll.cemantix.org/2012/data.html`

by the lines `"#begin nodes"` and `"#end nodes"` represents the `SpanSet` for this document as a sequence of one-lined `Span` descriptions. Each `Span` is defined by a 3-tuple of indices that indicate sentence-, begin- and end-index. The sentence-index is zero-based whereas the other two indices start at index 1 for the first word in a sentence. The notion of the generic `ROOT` node is optional. Each node besides this `ROOT` node can have an indefinite number of properties assigned to it after a separating tab-character (\t). Properties are defined as `"<key>:<value>"` statements with multiple properties separated by semicolon (;) and an optional extra semicolon after the last property.

Similar to the nodes section, the part listing edges is circumscribed by `"#begin edges"` and `"#end edges"` and orders its content one edge per line. The format of edges is such that both the source and target `Span` are given as defined in the nodes section with the special edge-separator `">>"` between them as in `"7-14-15>>26-2-3"`. Syntax for optional edge properties follows the exact same rules as outlined above for nodes.

A.2 Label Pattern Syntax

As shown in Chapter 4, the entity grid visualization supports a flexible way of customizing the grid labels. The following table lists all the characters that yield replacement strings when used within a label pattern:

Character	**Description**
\	Escaping character to allow for *magic characters* to be used without substitution
`b`	Begin-index of a `Span`
`e`	End-index of a `Span`
`c`	Number of `Spans` within the current sentence that share the given `Cluster`. This is exactly the default label when label patterns are deactivated
`r`	Range of the `Span`, i.e. the number of word tokens it covers, as given in the sentence structure
`l`	Length of the `Span` in terms of characters including white-spaces between word tokens

Table A.1: List of *magic characters* that can be used when defining a label pattern for the entity grid visualization.

Besides the magic characters listed above, there exist three special expressions to reference certain properties of a `Span` or the corresponding sentence. All of these property expressions are of the form `"x<key>x"` where `x` is the corresponding delimiter and `"<key>"` signals for which property to query:

Span-Properties '%...%' Returns the value of the specified property of the `Span`

Sentence-Properties '$...$' Returns a concatenation of the specified properties on the sentence level using all the words covered by the current `Span`. As example take the sentence *"John saw the green car ."* with word tokens stored in a property called `"form"` and a `Span` ranging from word 3 to 4. When the expression `"$word$"` is used the output for this label would be the sequence *"green car"*.

Head-Properties '§...§' As a specialized version of the *Sentence-Properties* expression this one does not concatenate the property values of all underlying word tokens but returns only the property for the index that is declared to be the `Span's` head. Using the above example sentence and `Span`, the output for the expression `"§word§"` would be *"car"*.

Each of these expressions result in a single hyphen (-) as output in the case that the desired property is not present at the respective object. All characters not introduced as magic characters or expression delimiters remain untouched by the pattern engine. There is no limitation on the size of a label pattern or the number of property expressions being used; however users are advised that extensive use of the latter can easily lead to confusing results since the space in each grid cell is limited unless manually widened.

Bibliography

Barzilay, R. and Lapata, M. (2008). Modeling local coherence: An entity-based approach. *Comput. Linguist.*, 34(1):1–34. (Cited on pages 13 and 26)

Baumann, S. and Riester, A. (2012). Referential and Lexical Givenness: Semantic, Prosodic and Cognitive Aspects. In Elordieta, G. and Prieto, P., editors, *Prosody and Meaning*, volume 25 of *Interface Explorations*, pages 119–162. Mouton de Gruyter, Berlin. (Cited on page 18)

Björkelund, A. and Farkas, R. (2012). Data-driven multilingual coreference resolution using resolver stacking. In *Joint Conference on EMNLP and CoNLL - Shared Task*, CoNLL '12, pages 49–55, Stroudsburg, PA, USA. Association for Computational Linguistics. (Cited on page 9)

Brants, S., Dipper, S., Hansen, S., Lezius, W., and Smith, G. (2002). The TIGER treebank. In *Proceedings of the workshop on treebanks and linguistic theories*, pages 24–41. (Cited on page 18)

Burchardt, A., Erk, K., Frank, A., Kowalski, A., and Pado, S. (2006). SALTO: A versatile multi-level annotation tool. In *Proceedings of LREC-2006*, Genoa, Italy. (Cited on page 18)

Eckart, K., Riester, A., and Schweitzer, K. (2012). A Discourse Information Radio News Database for Linguistic Analysis. In Chiarcos, C., Nordhoff, S., and Hellmann, S., editors, *Linked Data in Linguistics. Representing and Connecting Language Data and Language Metadata*, pages 65–76. Springer, Heidelberg. (Cited on pages 9 and 18)

Erk, K. and Padó, S. (2004). A powerful and versatile xml format for representing role-semantic annotation. In *LREC*. European Language Resources Association. (Cited on page 18)

Fernandes, E., dos Santos, C., and Milidiú, R. (2012). Latent structure perceptron with feature induction for unrestricted coreference resolution. In *Joint Conference on EMNLP and CoNLL - Shared Task*, pages 41–48, Jeju Island, Korea. Association for Computational Linguistics. (Cited on pages 9, 11 and 41)

Gärtner, M., Thiele, G., Seeker, W., Björkelund, A., and Kuhn, J. (2013). ICARUS – an extensible graphical search tool for dependency treebanks. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60, Sofia, Bulgaria. Association for Computational Linguistics. (Cited on pages 21 and 35)

Hana, J. and Štěpánek, J. (2012). Prague markup language framework. In *Proceedings of the Sixth Linguistic Annotation Workshop*, LAW VI '12, pages 12–21, Stroudsburg, PA, USA. Association for Computational Linguistics. (Cited on page 18)

Lezius, W. (2002). *Ein Suchwerkzeug für syntaktisch annotierte Textkorpora.* PhD thesis, IMS, University of Stuttgart. Arbeitspapiere des Instituts für Maschinelle Sprachverarbeitung (AIMS), volume 8, number 4. (Cited on page 18)

Mitkov, R. (2003). *The Oxford Handbook of Computational Linguistics (Oxford Handbooks in Linguistics S.).* Oxford University Press. (Cited on page 9)

Müller, C. and Strube, M. (2000). MMAX: a tool for the annotation of multi-modal corpora. In *Workshop on Adaptive Text Extraction and Mining - IJCAI 2001.* (Cited on page 17)

Müller, C. and Strube, M. (2006). Multi-level annotation of linguistic data with MMAX2. In Braun, S., Kohn, K., and Mukherjee, J., editors, *Corpus Technology and Language Pedagogy: New Resources, New Tools, New Methods*, pages 197–214. Peter Lang, Frankfurt a.M., Germany. (Cited on page 17)

Ng, V. (2010). Supervised noun phrase coreference research: The first fifteen years. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1396–1411, Uppsala, Sweden. Association for Computational Linguistics. (Cited on page 11)

Pajas, P. and Štěpánek, J. (2008). Recent advances in a feature-rich framework for treebank annotation. In *Proceedings of the 22nd International Conference on Computational Linguistics - Volume 1*, COLING '08, pages 673–680, Stroudsburg, PA, USA. Association for Computational Linguistics. (Cited on page 18)

Pajas, P. and Štěpánek, J. (2009). System for Querying Syntactically Annotated Corpora. In *Proceedings of the ACL-IJCNLP 2009 Software Demonstrations*, pages 33–36, Suntec, Singapore. Association for Computational Linguistics. (Cited on page 18)

Poláková, L., Jínová, P., Zikánová, Š., Bedřichová, Z., Mírovský, J., Rysová, M., Zdeňková, J., Pavlíková, V., and Hajičová, E. (2012). Manual for annotation of discourse relations in prague dependency treebank. Technical Report 47, Prague, Czech Republic. (Cited on page 9)

Pradhan, S., Moschitti, A., Xue, N., Uryupina, O., and Zhang, Y. (2012). CoNLL-2012 shared task: Modeling multilingual unrestricted coreference in OntoNotes. In *Proceedings of the Sixteenth Conference on Computational Natural Language Learning (CoNLL 2012)*, Jeju, Korea. (Cited on pages 9, 11, 23, 41 and 49)

Pradhan, S. S., Hovy, E., Marcus, M., Palmer, M., Ramshaw, L., and Weischedel, R. (2007). Ontonotes: A unified relational semantic representation. In *Proceedings of the International Conference on Semantic Computing*, ICSC '07, pages 517–526, Washington, DC, USA. IEEE Computer Society. (Cited on page 9)

Soon, W. M., Ng, H. T., and Lim, D. C. Y. (2001). A machine learning approach to coreference resolution of noun phrases. *Computational Linguistics*, 27(4):521–544. (Cited on pages 9 and 11)

Stenetorp, P., Pyysalo, S., Topić, G., Ohta, T., Ananiadou, S., and Tsujii, J. (2012). brat: a web-based tool for nlp-assisted text annotation. In *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 102–107, Avignon, France. Association for Computational Linguistics. (Cited on page 18)

Stenetorp, P., Topić, G., Pyysalo, S., Ohta, T., Kim, J.-D., and Tsujii, J. (2011). Bionlp shared task 2011: supporting resources. In *Proceedings of the BioNLP Shared Task 2011 Workshop*, BioNLP Shared Task '11, pages 112–120, Stroudsburg, PA, USA. Association for Computational Linguistics. (Cited on page 18)

Thompson, H. and McKelvie, D. (1997). Hyperlink semantics for standoff markup of read-only documents. In *Proceedings of SGML Europe'97*, Barcelona. (Cited on page 17)

Yimam, S. M., Gurevych, I., de Castilho, R. E., and Biemann, C. (2013). Webanno: A flexible,web-based and visually supported system for distributed annotations. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (System Demonstrations) (ACL 2013)*, pages 1–6, Stroudsburg, PA, USA. Association for Computational Linguistics. (Cited on page 19)

All links were last followed on September 26, 2013.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature