3 Architecture and Workflow

3.1 Tasks to Handle

To populate a database from text documents, we start from *(a)* unstructured information in a collection of potentially relevant texts and want to reach *(b)* structured information stored in a (most likely relational) database. How do we get from *(a)* to *(b)*, i.e., how do we identify relevant information in the texts and bring them into a suitable form for storing them in the database?

3.1.1 Prerequisites

To keep the scope of the system realistic, we assume that two tasks will already have been handled when the system starts its work:

Create target schema: The system needs a *target schema* to know which data to extract. Generally the relational schema of the target database might serve as target schema, or an underlying model (e.g. E/R model) might be used, but in many cases a simpler model will be sufficient—cf. Sec. 9.1 for the target schema used in this approach. For some IE algorithms, additional metadata might be required, or they might be able to make use of it if it is available. Assuming that a target database already exists, this task should usually be fairly trivial to handle.

Annotate sample texts: Information extraction is generally handled as a *supervised* learning process, due to the fact that it requires human judgment to decide which information should be extracted. The goal of IE is to learn a *model* that generalizes this human judgment sufficiently to automatically extract information from unknown texts of the same domain. Generally, the human judgment is provided in form of a set of sample texts manually annotated with the information to extract. While there are tools available for annotating texts in a comfortable fashion[1], this still is an extensive and burdensome process. For this reason, this work supports incremental training as an approach to reduce this burden by allowing to create the necessary human judgment in a more interactive way (cf. Sec. 3.3), but even this cannot remove the burden completely.

3.1.2 Tasks

In many cases, IE corpora will be "pure", i.e., they will comprise only texts that contain relevant information according to a single given target schema. However, especially in

[1] For example, the *XTract* tool developed by Heiko Kahmann for his diploma thesis [Kah03] under the supervision of Heinz Schweppe, Peter Siniakov, and the author.

more realistic settings it is possible that texts are *irrelevant* (they do not contain any relevant information) and/or that there are several target schemas for different types of texts. In such cases, texts can be *filtered* to find out which of them are relevant and to determine which target schema applies to which texts.

Typically, target schemas will contain attributes that are *explicitly* mentioned in a text (e.g. names, dates, geographic locations). These can be extracted by determining and extracting suitable text fragments. Depending on the attribute type in the target database, it might be necessary to *normalize* values to fit the attribute domain and allow easier querying. *Value normalization* will be mainly a type-specific task (specialized rules and heuristics for dates, person names, geographic entities etc.), that will most likely be rule-based.

While IE does not try to aim at text understanding, in some cases it might be both useful and possible to extract *implicit* information. For example, the *Interface* of a software product could be a *"GUI"*, *"command-line"* or *"Web interface"*, or the topic area of a seminar might be picked among a list of predefined values. Such information often will not be expressed explicitly, but it might be determined by techniques such as text classification as long as an enumeration of possible values is specified in the target schema and suitable training data exists.

In some tasks, the combination of extracted attributes into relational tuples is trivial: either there is only a single relation and each text contains one tuple so all attributes extracted from a text can be stored in the same tuple—if there are several candidate values for an attribute, typically all but the most likely one are discarded. This setup is assumed by many standard tasks such as the *Seminar Announcements* and *Corporate Acquisitions* from the *RISE Repository* [RISa] (cf. Chap. 17). Alternatively, attributes are completely independent of each other and each relation comprises only a single attribute—this is frequently the case in named entity and bioinformatics tasks. In cases of other, more complicated target schemas, it is necessary to handle *relationship resolution* (also called *template unification*) between extracted attributes to combine them into appropriate tuples (if a text can contain several tuples of the same relation or of different relations) and/or to resolve dependencies (key constraints) between relations.

Some target schemas might define additional constraints, e.g., semantic constraints (the start time of a seminar must be smaller than its end time, the number of participants must not be higher the room capacity). Semantic constraints affecting several attributes have to be treated as part of the *relationship resolution* task, while constraints for a single attribute ("*CHECK (VALUE IN ...)*") should be considered during *value normalization*.

In many cases, information given in various texts might refer to the same real-world entity, e.g., a *seminar* might be mentioned in two different texts. The goal of *instance unification* is to find out where this is the case and merge complementary or conflicting pieces of information. This task is especially challenging if there are conflicts or if preexisting information should be updated, e.g., if the room of a seminar has been changed—temporal databases could be used to keep track of changes. In the database world, *instance unification* is known as *record linkage* or related terms, but it might

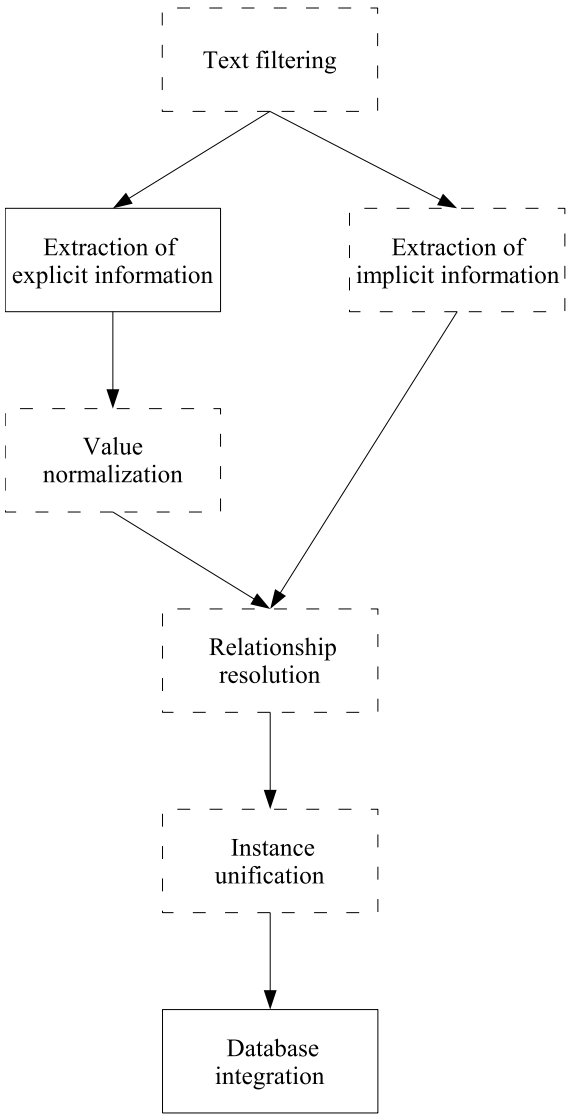

Figure 3.1: Tasks to Be Handled

be useful to handle this task in the context of information extraction since input texts contain additional information that will no longer be available after the extracted facts have been stored in a database.

After all these steps, the extracted data should be ready for insertion into the target database.

The comprehensive lists of tasks to be handled by a full IE system thus looks as shown in Fig. 3.1. Most of the tasks are optional—these are enclosed in dashed boxes. Handling of explicit and implicit information can be performed in parallel.

3.2 Architecture of a Typical IE System

So far we have discussed which tasks there are to handle, but not how they are handled. Typical trainable IE systems follow a pipeline architecture that comprises a

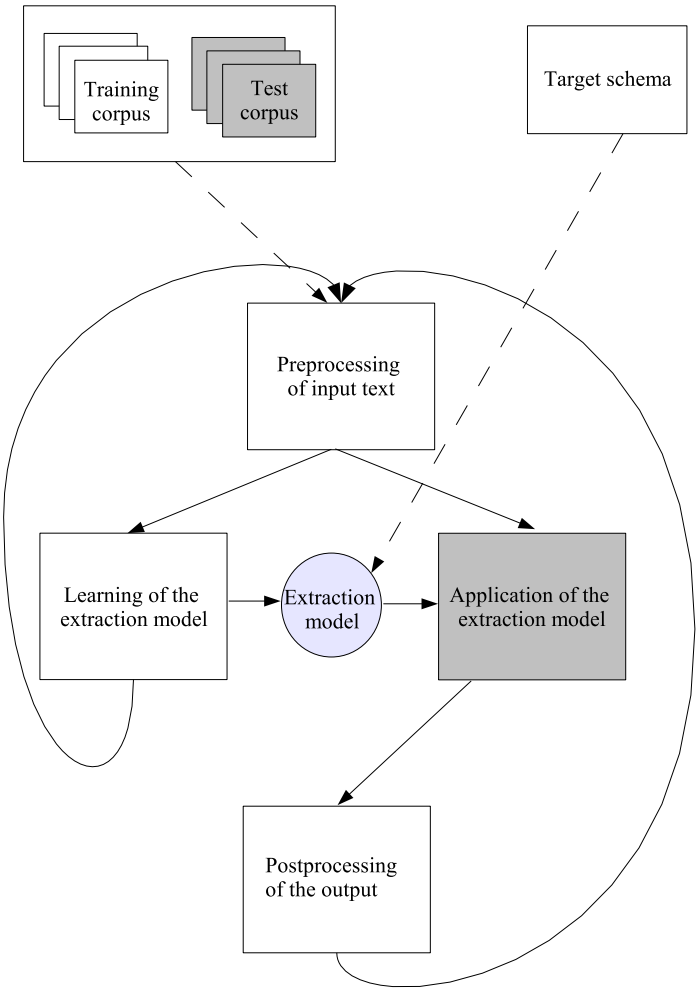

Figure 3.2: Architecture of a Typical IE System

static preprocessing stage, an adaptive learning and application stage and, during the application phase, a static postprocessing stage as the three main blocks (Fig. 3.2). Each of them handles a subset of steps that are particularly relevant for a pursued approach.

A text corpus including texts of the application domain and a target schema defining what the relevant information is constitute the minimum input for an IE system. Besides, it can be supported by additional semantic resources provided by a human.

Preprocessing of Input Texts: Text corpora often consist of unstructured, "raw" natural language texts. A big part of the relevant information can be distinguished by some regularity found in the linguistic properties of texts. Thus linguistic analysis can give helpful hints and determine important features for identifying relevant content. Regarding the tasks described in the previous section, this step will usually occur after text filtering (to avoid the unnecessary preprocessing of irrelevant texts) but before all other tasks.

The following linguistic components proved to be useful for information extraction:

Tokenization: Starting with a sequence of characters the goal is to identify the elementary parts of natural language: words, punctuation marks and separators. The resulting sequence of meaningful tokens is a base for further processing.

Sentence Splitting: Sentences are one of the most important elements of the natural language for structured representation of the written content. Binding interrelated information, they are the smallest units for expression of completed thoughts or events. The correct recognition of the sentence borders is therefore crucial for many IE approaches. The task would be trivial if the punctuation marks were not ambiguously used. Correct representation of a text as a sequence of sentences is utilized for syntactic parsing.

Morphological Analysis: Certain facts are typically expressed by certain parts of speech (e.g. names). Determining parts of speech of tokens is known as POS tagging. Statistical systems can use POS tags as classification features, rule-based systems as elements of extraction rules. Segmentation of compounds, recognition of flection forms and consecutive normalization disclose further important morphological features.

(Chunk) Parsing: While full sentence parsing is preferred by knowledge-based systems, some statistical approaches rely on chunk parsing—shallow syntactic analysis of the sentence fragments performed on phrasal level. It is justified by the fact that the extracted information is often completely included in a noun, verb or prepositional phrase that comprises the most relevant context for its recognition.

Named Entity Recognition, Coreference Resolution Named entities are one of the most often extracted types of tokens. Some approaches use a simple lookup in predefined lists (e.g. of geographic locations, company names), some utilize trainable Hidden Markov Models to identify named entities and their type. Coreference resolution finds multiple references to the same object in a text. This is especially important because relevant content may be expressed by pronouns and designators ("she held a seminar", "The company announced"). Both tasks require deeper semantic analysis and are not as reliable as other linguistic components. They are only occasionally used be rule-learning approaches and almost never be statistical ones.

While for knowledge-based and some rule-based systems, linguistic preprocessing is an element of the core system, for statistical and other rule-based approaches it is optional but can have a serious impact on the quality of extraction—we will analyze the impact of linguistic preprocessing on our own system as part of the in the ablation study presented in Section 18.1.

This stage will usually be performed after the *text filtering* task mentioned in Section 3.1 (if used) but prior to any other tasks.

Learning and Application of the Extraction Model: The application range of today's IE systems is intended to be as wide as possible. The features of a concrete domain cannot be hardwired in a system since the adaptation effort to other domains is too high. Modern systems use a learning component to reduce the dependence on specific domains and to decrease the amount of resources provided by human. An extraction model is defined according to the pursued approach and its parameters are "learned" (optimized) by a learning procedure. In case of statistical approaches, the extraction model typically comprises a way of modeling the information extraction task that makes it accessible for statistical methods (e.g., by treating the information to extract as the hidden state and the processed text as the visible output of a Hidden Markov Model, cf. Sec. 4.2) and defining a set of available features. The learning processing then consists in optimizing the non-fixed parameters of the chosen model (e.g., the transition and output probabilities of the HMM). Rule-based approaches model the task in a way that allows learning and applying a set or list of extraction rules and define a set of features that can be used in rules and of way of creating extraction rules from training examples. Knowledge-based approaches acquire structures to augment and interpret their knowledge for extraction. The general challenge is to find an extraction model that allows learning all relevant domain parameters using the same extraction framework for each application domain.

Considering the problems and complexity of IE, *supervised learning* appears to be the most appropriate and is the most widely used learning mode. The majority of approaches prefer annotated training corpora albeit some rely on human supervision during the learning stage. To assess the quality of an approach the training text corpus is created by annotating text fragments that contain relevant content and divided into two parts. One part, the training set, is used for training (learning the parameters of the extraction model) and another, the test set, is used to test the ability of the model to correctly extract new information it was not trained on. The test results can also be used to improve the extraction model to perform better on new domain texts when applied to real domain texts.

Some approaches allow further refinement of an extraction model based on the human feedback about extractions during the application. The newly evaluated extractions can be incorporated as new training instances and the model can be retrained.

The learning component is crucial for an IE system, because it comprises the algorithms for identification of relevant text parts and transferring them according to the target schema. This stage comprises the extraction of explicit and implicit information, the core of each IE system.

Postprocessing and Integration: After the relevant information has been found by application of the extraction model, the identified text fragments are assigned to the corresponding attributes of the target schema. They can be normalized according to the expected format (e.g. representation of dates and numbers). Some facts may appear in the input texts more than once or already exist in

the database. In this case, different instances could be merged (instance unification). Finally, the identified, normalized and unified information is stored at the appropriate relation in the database.

This stage thus comprises the remaining tasks from Section 3.1, starting with *value normalization* (if used). Most current trainable systems IE do not yet perform much postprocessing, leaving such tasks as future work.

3.3 Active Learning and Incremental Learning

As stated above (Sec. 3.1.1), the annotation of a sufficient amount of training data is the primary burden that we have to deal with when adapting a trainable IE system to a new domain. This is already a large progress compared to the "classical" static rule-based IE systems which required a manual rewriting of the rules used in the system (a time-consuming and intricate task that must be done by experts which are usually hard to get), but it still requires a considerable amount of work.

To address this, some approaches use *active learning* [Fin03, Sch02] where the system actively selects texts to be annotated by a user from a pool of unannotated training data. Thus adaptation to a new domain still requires a large amount of raw (unannotated) training data (which are usually cheap), but only a reduced amount of annotated (and thus expensive) training data which are chosen to be especially valuable for building the extraction model, e.g., those texts or fragments whose annotation is least certain.

An alternative setup is *incremental learning* (also called *incremental training*). Here training documents are annotated sequentially by a user and immediately incorporated into the extraction model. Except for the very first document(s), the system can support the user by proposing attribute values. Thus the work to be done by the user is reduced over time, from largely manual annotation of attribute values to mere supervision and correction of the system's suggestions.

While incremental learning generally requires more annotated documents than active learning to reach the same level of accuracy (since the system cannot select the most informative samples), the work required by the user for annotating each document is reduced. Also the user keeps control about which documents are processed. Moreover an incremental setup fits better in situations where information is to be extracted from a stream of incoming documents ("text stream management"), for example e-mail messages or newspaper articles.

3.4 Workflow

We will demonstrate the resulting working on an example scenario where the task is to extract information from a stream of incoming e-mail messages. Such a task is most appropriately handled by *incremental learning*, since there is no pre-existing corpus from which to select training and test sets, and the typical content of documents might change over time.

Figure 3.3: Sample Interface: Information Extraction from E-Mail Messages

In an incremental setup, the workflow will comprise all or some of the following steps:

1. Filter potentially relevant vs. irrelevant documents, e.g., spam (junk) e-mail vs. non-spam messages. This is a binary (two-class) text classification task. This step is unnecessary if the corpus is known to contain only relevant documents. It will only work if the two classes of documents are sufficiently heterogeneous to allow the acquisitions of a classification model suitable for separating them.
2. Determine the type of a document based on the existing templates (whether it is a *Seminar Announcement* or a *Job Application* etc.). This is a multi-class text classification task. This step is unnecessary if the types of all documents are known or if there is only a single type of documents in the corpus.
3. Fill the attributes defined by the selected target schema, extracting relevant *explicit* and possibly *implicit* information. Our approach for extracting explicit information will be introduced in Chap. 10. The extraction of implicit information (such as the topic area of a seminar) could again be handled as a multi-class text classification task.
4. Perform any postprocessing steps such as value normalization, relationship resolution and instance unification, if these steps are handled.
5. Show the predicted information to the user; ask the user to review the information and to correct any errors and omissions. This allows the user to quickly capture (and possible store) relevant pieces of information from the received message. Further actions can be defined depending on the used templates, for example the system could offer the user to add a *Seminar Announcement* to her calendar and to notify her when the lecture is about to start. An example of a possible user interface is outlined in Fig. 3.3.

6. Retrain the classification models based on the user's feedback.

If batch training is used instead of incremental training, the last step is omitted.