information extraction
automatic markup
macro-level markup
micro-level markup

Information Extraction and Automatic Markup for XML documents

Mohammad Abolhassani[1], Norbert Fuhr[2], and Norbert Gövert[3]

[1] University of Duisburg-Essen mohasani@is.informatik.uni-duisburg.de
[2] University of Duisburg-Essen fuhr@uni-duisburg.de
[3] University of Dortmund goevert@ls6.cs.uni-dortmund.de

1 Introduction

As XML is going to become the standard document format, there is still the legacy problem of large amounts of text (written in the past as well as today) that are not available in this format. In order to exploit the benefits of XML, these legacy texts must be converted into XML. In this chapter, we discuss the issues of automatic XML markup of documents. We give a survey on existing approaches, and we describe a specific system in some detail.

When talking about XML markup, we can roughly distinguish between three types of markup:

- *Macro-level markup* deals with the global visual and logicalstructure of a document (e.g. part, chapter, section down to the paragraph level.)
- *Micro-level markup* is used for marking single words or word groups. For example, in news, person and company names, locations and dates may be marked up, possibly along with their roles in the event described (e.g. a company merger).
- *Symbol-level markup* uses symbolic names as content of specific elements in order to describe content that is not plain text (e.g. MathML for mathematical formulas and CML for chemical formulas). Since this type of content is usually represented in various formats in legacy documents, specific transformation routines should be applied in order to convert these into XML. We will not consider this type of markup in the remainder of this chapter.

Micro and macro-level markup require different methods for performing automatic markup: Whereas macro-level markup is mainly based on information about the layout of a document, micro-level markup typically requires basic

macro-level markup

linguistic procedures in combination with application-specific knowledge. We will describe the details of these two approaches in the following sections.

Adding markup to a document increases its value by making its information more accessible. Without markup, from a system's point of view, a document is just a long sequence of words, and thus the set of operations that can be performed on such a document is rather limited. Once we have markup, however, the system is able to exploit the implicit semantics of the markup tags, thus allowing for operations that are closer to the semantic level. Here we give a few examples:

- Markup at the macro level supports a user in navigating through the logical structure of a document.
- Content-oriented retrieval aims at retrieving meaningful units for a given query that refers only to the content, but not to the structure of the target elements. Whereas classical passage retrieval [1] can only select text passages of a fixed size, XML-based retrieval is able to select XML elements based on the explicit logical structure as represented in the macro-level markup.
- When micro-level markup is used for specifying the data type of element content (e.g. date, location, person name, company name), type-specific search predicates may be used in retrieval, thus supporting high-precision searches.
- Another dimension of micro-level markup is the role of element content (e.g. author vs. editor, departure location vs. arrival location, starting date vs. ending date). Here again, precision of retrieval can be increased by referring to these elements; also, browsing through the values occurring in certain roles may ease information access for a user.
- Text mining can extract the contents of specific elements and stores them in a separate database in order to perform data-mining-like analysis operations.

The remainder of this chapter is structured as follows: In Section 2, we briefly describe methods for macro-level markup. In Section 3, we give a survey over Information Extraction (IE) methods and discuss their application for micro-level markup. In Section 4, we present a case study where a toolkit for automatic markup, developed by our research group, is applied to articles from encyclopedias of art. Finally, we conclude this chapter with some remarks.

2 Markup of Macro Structures

In case the original document is available in the electronic format, markup of the macro structure of the document is a matter of conversion, translation or transformation from the original structure to the target (XML) elements. Otherwise, providing an electronic representation of the document is the prerequisite of any markup at both macro and micro level.

In order to get an electronic representation, the original document must be processed. *Optical Character Recognition (OCR)* is the process of converting text from paper into a form that computers can manipulate, by scanning the paper, producing image, analysing it and providing an electronic file.

In this way, the textual content of the document as well as its structure can be extracted. The document structure can be expressed in two ways:

- Presentation oriented: how the document looks.
- Logically: how the document parts are related to each other.

Markup of these (macro) structures have different applications. Presentation markup can mainly be used for enhancing the layout of the document. Southall mentions that presentation markup helps us to display a document's visual structure which contributes to the document's meaning [2]. Logical markup serves for a variety of purposes, as mentioned in the previous section.

Taghva et al. introduce a system that automatically markups technical documents, based on information provided by an (OCR) device, which, in addition to its main task, provides detailed information about page layout, word geometry and font usage [3]. An automatic markup program uses this information, combined with dictionary lookup and content analysis, to identify structural components of the text. These include the document title, author information, abstract, sections, section titles, paragraphs, sentences and dehyphenated words.

Moreover, the logical structure of a document can be extracted from its layout. For this purpose, there are two approaches:

Top-down: starting with the presentation markup and joining segmented pages into sections, sections into paragraphs, paragraphs into sentences and sentences into words. This is the preferred approach in the literature.

Bottom-up: starting with words and grouping words into sentences, sentences to paragraphs and paragraphs into sections.

Having detailed information about presentation attributes of single words (its page, exact location on the page and font) one can use this data to form sentences, paragraphs and sections [3].

Furthermore, Hitz et al. advocate the use of synthetic document images as a basis for extracting the logical structure of a document, in order to deal with different formats and document models [4].

3 Survey on Information Extraction

The task of information extraction can be considered as a problem of template filling: For a given domain, users are interested in facts of a certain type (e.g. joint ventures, management changes), which are to be extracted from a collection of texts. For this purpose, a template form has to be defined, and the system is supposed to fill it. Accordingly, the system has to scan large volumes

knowledge engineering
machine learning

of text and instantiate the template for each event of the specified type, by filling it with the data extracted from the text. Thus, we can assert that Information Extraction (IE) is more challenging than Information Retrieval (IR): whereas the latter only aims at locating the relevant documents for a given query, the former goes one step further, by extracting facts from the text of the relevant documents. Since in IE, as in IR, systems cannot yield perfect results for a given task, the quality of the result is a crucial issue. For this purpose, standard IR measures like recall, precision and the F-measure are applied for IE as well. Evaluation of IE systems have been the major objective of the Message Understanding Conferences (MUC) [5], where participating groups were competing in developing systems for a given task. The MUC proceedings also give a good overview of the range of approaches tried.

In the following, we give a brief survey of the state of the art in information extraction. For a broader and more detailed description, see e.g. [6, 7, 8].

3.1 Approaches for Building IE Systems

For building information extraction systems, there are two generic types of approaches:

Knowledge engineering approach: Here the rules to be applied by the system are constructed manually by a so-called knowledge engineer. This strategy may be applied to the construction of the grammar rules as well as to the discovery and formulation of the domain patterns. Of course, effective rule sets can be constructed only iteratively, by starting with simple rule sets, evaluating the results and refining them stepwise. Thus the knowledge engineering approach is rather laborious.

Machine learning approach: To apply this approach, also called automatic training, we need annotated corpora. For example, a name recogniser would be trained by annotating a corpus of texts with the domain-relevant proper names. Alternatively, training data can be obtained by close interaction with the user, where the system proposes new rules, which are either confirmed or rejected by the user. For learning the rules, statistical methods are used.

These two types of approaches can be applied at various stages of the IE process (see below). Also, it is possible to mix the strategies, e.g. one can use machine learning for named entity recognition and knowledge engineering for formulating the domain patterns. Generally speaking, knowledge engineering should be preferred in case appropriate knowledge (e.g. lexicons) and human resources (i.e. rule writers) are available, whereas the machine learning approach should be preferred when the required knowledge and resources are missing, but enough training data is available. In terms of achievable performance, although the former approach has advantages, usually the latter approach also leads to good results.

3.2 System Architecture

morphological processing
lexical processing

A typical IE system consists of the following four major blocks:

1. *Tokenisation* performs word segmentation, which is trivial for European languages (words are separated by blanks and / or punctuation symbols), whereas some Asian languages require sophisticated methods for mapping a text into a sequence of words.
2. *Morphological and lexical processing* deals with the recognition of inflected word forms.
3. *Syntactic analysis* is applied to the output of the previous stage, and can be shallow or aim at full parsing.
4. *Domain analysis* finally fills the specified template based on the input from the previous stages.

In the following, we describe the last three stages in more detail.

Morphological and Lexical Processing

Following tokenisation, the system first has to detect inflectional variants of word forms. For some languages, e.g. English and Japanese, this task can be accomplished by simple string processing rules, whereas languages with a complex morphology, e.g. German and Finnish, usually require a lexicon in order to map the inflected word forms to the non-inflected ones. Furthermore, during this step compound words should be segmented into their components, which, again, requires a lexicon.

Following the morphological analysis, a lexical lookup retrieves syntactical and / or semantical information for the given words. However, many application domains use a specific sub language which is hardly covered by a standard lexicon. Thus, most approaches use small, domain specific lexicons. In any case, lexical coverage will always be limited, thus the system design has to take lexical incompleteness into account.

In order to ease subsequent syntactic analysis, most systems perform part of speech tagging. Simple approaches just collect the unigram tag frequencies of a word and then apply a rareness threshold, whereas other methods also take into account bigram frequencies in order to reduce the ambiguity of word senses.

The most important task in the morphological and lexical component is the recognition of names (e.g. person, product or company names, locations) and structured entities (e.g. dates, times), which are typically too numerous to be included in a lexicon. Whereas elements of the latter group can often be recognised by means of simple regular expressions, the elements of the former group require the development of appropriate recognisers, using either the knowledge engineering or the machine learning approach. Most name recognition systems use the latter strategy in combination with Hidden Markov Models. However, large training corpora (> 100 K words) are needed in order to achieve a high performance, and increasing the training data size only leads to log-linear improvements.

syntactic analysis
domain analysis

Syntactic Analysis

The most natural approach to syntactic analysis would be the development of a full parser. However, experiments have shown that such an approach results in a very slow system, which is also error-prone. Thus, most approaches in this area aim at shallow parsing, using a finite-state grammar. The justification for this strategy lies in the fact that IE is directed toward extracting relatively simple relationships among singular objects. These finite-state grammars focus mainly on noun groups and verb groups, since they contain most of the relevant information. As attributes of these constituents, numbers and definiteness are extracted from the determiner of noun groups, and tense and voice from verb groups. In a second parsing phase, prepositional phrases are handled; here mainly the prepositions "of" and "for" are considered, whereas treatment of temporal and locative adjuncts is postponed to the domain analysis phase.

Domain Analysis

Before the extraction of facts can start, first the problem of coreference must be solved. Since text writers typically use varying notations for referring to the same entity, IE systems struggle with the problem of resolving these coreferences (e.g. "IBM", "International Business Machines", "Big Blue", "The Armonk-based company"). Even person names already pose severe problems (e.g. "William H. Gates", "Mr. Gates", "William Gates", "Bill Gates", "Mr. Bill H. Gates"). In addition, anaphoric references (pronouns or discourse definite references) must be resolved. Although there is rich literature on this specific problem, most approaches assume full parsing and thus are not applicable for IE.

In [6], a general knowledge engineering approach for coreference is described: In the first step, for a candidate referring expression (noun phrase), the following attributes are determined: sortal information (e.g. company vs. location), number (single vs. plural), gender and syntactic features (e.g. name, pronoun, definite vs. indefinite). Then, for each candidate referring expression, the accessible antecedents are determined (e.g. for names the entire preceding text, for pronouns only a small part of it), which are subsequently filtered with a semantic / sortal consistency check (based on the attributes determined in the first step), and the remaining candidates are filtered by dynamic syntactic preferences (considering the relative location in the text).

Once there are solutions for all the problems described above, the core task of IE can be addressed. As a prerequisite, an appropriate template form must be defined: Typically, users would give an informal specification of the information bits they are interested, for which then an adequate and useful representation format must be specified.

For filling this template, there are two knowledge engineering approaches:

- The *molecular* approach aims at filling the complete template in one step. For this purpose, the knowledge engineer reads some texts in order to

identify the most common and most reliably indicative patterns in which relevant information is expressed. For these patterns appropriate rules are formulated, then one moves on to less common but still reliable patterns. Thus, this approach aims initially at high precision, and then improves recall incrementally.

- The *atomic* approach, in contrast, is based on the assumption that every noun phrase of the right sort and every verb of the right type (independently of the syntactic relations among them) indicates an event / relationship of interest. Thus, one starts with high recall and low precision, with incremental development of filters for false positives. This approach is only feasible if entities in the domain have easily determined types, and there is scarcely more than one template slot where an entity of a given type may fit - as a negative example, a template for management changes would contain at least two slots (predecessor / successor) where a person can be filled in.

In most cases both approaches produce only partial descriptions, which must be merged subsequently (e.g. in a template for management changes, one portion of the text may mention company, position and the new person filling it, whereas the predecessor is mentioned in a subsequent paragraph). This merging step is a specific type of unification. A knowledge engineering approach for this problem is described in [6]: Starting from typed slots, type-specific procedures are developed which compare two candidates for inconsistencies, coreference and subsumption; in addition, application-specific heuristics are necessary in most cases.

In general, major parts of an IE system are rather application-dependent, and there is little experience with the development of portable systems. On the other hand, experience from the MUC conference shows that approaches based on general-purpose language analysis systems yield lower performance than application-specific developments.

3.3 Types of IE Problems

Above, we have described the general structure of IE tasks and the architecture of IE systems. A more detailed analysis and categorisation of IE problems is described in [9]. In this paper, the authors distinguish between source properties and extraction methods, and develop taxonomies for issues related to these two subjects.

With respect to the source properties, the following aspects are considered to be the more important ones:

- *Structure* can be free, tagged or even follow a specified schema.
- *Topology* distinguishes between single and multiple documents to be considered for filling a single template.
- *Correctness* refers to the amount and type (format, content) of errors that may occur in the input.

named entity recognition
coreference resolution

- *Regularity* specifies whether instances of source documents may be regular in format or vary widely.
- *Stability* characterises type and frequency of source documents with respect to content and structure.
- *Interaction* specifies the ways in which a source notifies an IE system about data changes - which may invalidate the current extraction description) - possibly along with a description of the changes.

Extraction methods may be characterised by the following features:

- The degree of *automation*, ranging from manual (programming by hand or by demonstration) to (semi-)automatic, where the type of the learning approach and the seed structure are important characteristics.
- The *extraction engine* may be based on a finite state automaton or a context-free grammar. Alternatively, if the source data already conforms to a certain data model, a query engine based on that model can be applied. Finally, some systems follow none of these approaches and use procedural descriptions instead.
- *Change and error handling* describes the ways in which the extraction engine copes with these problems. Without additional provisions, the system might fail; it should at least detect changes and give an appropriate warning. Some systems are able to compensate for errors to a certain extent. A detect and learn strategy would even go one step further and try to adapt the system dynamically.
- *Use of source knowledge* simplifies the rule set for IE. This knowledge may be at the syntactical as well as semantic level.
- *Use of target knowledge* refers to the fact that both syntax (e.g. free text vs. record structure) and semantics (e.g. predefined list of terms) of the desired output may be restricted
- Finally, *transformation capabilities* describe the level of transformation, which may be at the structural or the semantic level.

3.4 Information Extraction for Automatic Markup

The discussion above has focused on IE methods, and little has been said about their relationship with the problem of *Automatic Markup (AM)*. Cunningham distinguishes five levels of IE tasks, which can also be used for characterising different levels of automatic markup [8]:

- *Named entity recognition* extracts entities of one or more given types. For automatic markup, this method can be used for assigning appropriate tags to these names as they occur in the text.
- *Coreference resolution* recognises different notations for the same entity. In XML, this fact could be marked by adding ID / IDREF attributes to the tags.

- *Template element reconstruction* deals with structured entities, where entity names and attributes are coordinated. In terms of XML, this would correspond to a structured element.

template element reconstruction

- *Template relation reconstruction* requires the identification of the relation between the template elements, e.g. an employee relationship between a company and a person. For automatic markup, this step would result in assigning additional tags characterising the roles of entities in such a (binary) relationship

template relation reconstruction

- *Scenario template production* finally refers to the filling of the complete template (including the merger process). This corresponds to automatic markup for DTD describing the relevant aspects of the application.

scenario template production

Following the machine learning approach, a number of algorithms specifically designed for automatic markup have been described in the literature.

MarkitUp! [10] is an early predecessor of automatic markup systems. It uses regular expressions and uses a kind of inductive logic programming for abstracting from the given examples. In addition, rules may be specified explicitly by the user, and the system combines both types of rules in a single grammar.

RAPIER [11] is a system for semi-structured text that uses a form of inductive logic programming for inferring rules from a corpus tagged with target templates. Here each slot filler has three fields: the target field, the tokens preceding the target phrase, and those following it. Rapier considers lexical, semantical and morphological constraints.

WHISK [12] is a rather general rule extraction system using regular expressions as extraction patterns. When combined with a parser, the system can also perform free text analysis. For learning, WHISK uses a covering algorithm inducing rules top-down.

SRV [13] considers all possible phrases as potential slot fillers. A multi-strategy approach combines evidence from three classifiers (rote learner, naive Bayes classifier, relational rule learner).

LearningPINOCCHIO [14] is based on a covering algorithm that learns rules by bottom-up generalisation of instances in a tagged corpus. Unlike other systems, LearningPINOCCHIO recognises starting tags and ending tags separately, which allows for easier generalisation in rule writing.

Vasari, Giorgio
knowledge engineering

4 The Vasari Project

Vasari[4] aims at developing a Web portal providing comprehensive knowledge about art, artists and works of arts for scientists and experts as well as for other interested people.

The starting point for the project are different encyclopedias of art in German which have been scanned and processed with an OCR software. The result is a collection of fairly unstructured plain texts representing the contents of the encyclopedias. An example of how such an texts like is given in Figure 1[5]

The task of the Vasari project is to develop means for doing automatic markup of the knowledge contained in these texts at the micro level, for providing search and navigational structures that allow for effective exploration of the knowledge.

Da Vinci, Leonardo, born in Anchiano, near Vinci, 15 April 1452, died in Amboise, near Tours, 2 May 1519. Italian painter, sculptor, architect, designer, theorist, engineer and scientist. He was the founding father of what is called the High Renaissance style and exercised an enormous influence on contemporary and later artists. His writings on art helped establish the ideals of representation and expression that were to dominate European academies for the next 400 years. The standards he set in figure draughtsmanship, handling of space, depiction of light and shade, representation of landscape, evocation of character and techniques of narrative radically transformed the range of art. A number of his inventions in architecture and in various fields of decoration entered the general currency of 16th-century design.

Fig. 1. Example text of source documents

To arrive at a micro-level markup from the OCRed texts, the Vasari project follows the knowledge engineering approach. The project presents a language and a number of tools for this purpose. Rules for markup can be expressed in the *Vasari Language (VaLa)*. The *Vasari tool* serves the knowledge engineer in the iterative process of developing the rules. Having defined the set of rules for a given encyclopedia, the extraction tool *VaLaEx (Vasari Language Extractor)* uses them in order to automatically markup the plain texts. Furthermore, a toolkit has been specified around VaLaEx which includes tools that can be used to pre-/post-process the input/output of the VaLaEx extractor.

In the following we describe the Vasari project in more detail. In Section 4.1 we give a survey on VaLa. The Vasari tool for developing VaLa descriptions

[4] Giorgio Vasari, who lived in the 16th century in Florence, Italy, was an Italian painter and architect. His most important work is an encyclopedia of artist biographies ("The biographies of the most famous architects, painters and sculptors", published in an extended edition in 1568), which still belongs to the foundations of art history.

[5] Although our work principally deals with German texts, here we give an English example from *The Grove Dictionary of Art Online* (`http://www.groveart.com/`).

and the VaLaEx markup tool are described in Section 4.2. In Section 4.3 we show how additional tools can be applied to enhance the result of the markup process. We end the description of the Vasari project in Section 4.4 with a brief discussion on how the results can be improved by fusing knowledge obtained from different sources.

XML Schema
regular expressions
linguistic categories

4.1 VaLa: The Vasari Language

Given a set of rather unstructured plain text documents, an appropriate description language for automatic markup must fulfill two requirements:

1. The logical structure of the documents implicitly inherent in the source documents must be made explicit within a description for automatic markup. Such a structure description defines the template which is to be filled through the markup process.
2. Given a source document, it must be stated how its contents can be mapped onto the elements of the template.

XML Schema [16] served as a starting point for VaLa. In our application however, XML Schema does not serve its original purpose to validate the structure of a given XML document, but to create such XML documents. The first requirement mentioned above is met by XML Schema directly. Structure can be defined in an elaborated way [17]. Elements within the structure can be further constrained by means of already built-in and extensible data types [18].

The second requirement can be accomplished for by the application specific data, which can be specified within `xsd:annotation` elements within XML Schemas. The VaLa extension to XML Schema contains means to specify filler rules defined within XML Schema. The filler rules define how text from the source documents are to be mapped onto the elements of the structure defined within the XML Schema. A filler rule may consist of up to three parts:

- Within the `<vala:match>` element constraints for the part of the source document - which is to be mapped into a given element - can be specified.
- The content of the `<vala:pre>` element describes constraints for the left context of a match for a given element.
- The content of the `<vala:post>` element describes constraints for the right context of a match for a given element.

In order to formulate the constraints within the filler rules, the following types of linguistic knowledge can be exploited:

- Character strings within source documents can be matched by *regular expressions*. The construction of complex regular expressions is facilitated by the option of nesting arbitrary expressions.
- Linguistic constructs like words and word classes can be matched by *computer linguistic categories*. For extracting such categories from the source documents external tools e.g. SPPC [19] can be used.

- Special types of entities (e.g. place names) can be matched by means of *registries* or *dictionaries* (e.g. gazetteers).

Figure 2 displays an excerpt from a VaLa description. Here, regular expressions are used for automatic markup of the artists' names. A name consists of two parts, a surname, followed by a given name. Both parts of the name start with an upper case letter, followed by at least one lower case letter (`[A-Z][a-z]+`). They may consist of more than one word separated by blanks, and are followed by a comma. A birth place is constrained by its left context which must consist of "born" followed by "in", with possible blanks and line feeds between and after them.

```
<xs:simpleType name="tSurName">
  <xs:annotation>
    <xs:appinfo>
      <match type="regexp">[[A-Z][a-z]+ ]+</match>
      <post type="regexp">,</post>
    </xs:appinfo>
  </xs:annotation>
</xs:simpleType>

<xs:simpleType name="tGivenName">
  <xs:annotation>
    <xs:appinfo>
      <match type="regexp">[[A-Z][a-z]+ ]+</match>
      <post type="regexp">,</post>
    </xs:appinfo>
  </xs:annotation>
</xs:simpleType>

<xs:simpleType name="tBirthPlace">
  <xs:annotation>
    <xs:appinfo>
      <pre type="regexp">born[ ]*[/n]*in[ ]*[/n]*</pre>
      <match type="regexp">[^,]+</match>
    </xs:appinfo>
  </xs:annotation>
  </xs:simpleType>
</xs:simpleType>
```

Fig. 2. An excerpt from a VaLa description for automatic markup of the artists' data

Given a VaLa description for documents of a special type (in our case articles from an encyclopedia) the *VaLaEx* tool for automatic markup applies that description onto a set of source documents. Since a VaLa description defines a tree structure, the approach taken by VaLaEx is based on the recursive definition of trees. The source document is passed to the root of the

structure tree. By means of its filler rules the root node selects that part of the document text which matches its filler rules. The matching part then is passed to the root's first child node, which in turn selects its matching text part and provides the respective markup. The remaining text is passed to the next child node, and so on. In case the filler rules within a child node cannot be matched, alternative solutions are tried through backtracking. For each child which receives text from its parent node, the algorithm is applied recursively.

Figure 3 displays the result of a VaLa description (the one mentioned above).

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Artist>
  <Name>
    <SurName>Da Vinci</SurName>,<GivenName> Leonardo</GivenName>
  </Name>, born in <MasterData>
    <BirthPlace>Anchiano</BirthPlace>, near Vinci, <BirthDate>
      <BirthDay>15</BirthDay> <BirthMonth>April</BirthMonth>
  <BirthYear>1452
  </BirthYear></BirthDate>, died
  in <DeathPlace>Amboise</DeathPlace>, near Tours, <DeathDate>
      <DeathDay>2</DeathDay> <DeathMonth>May</DeathMonth>
  <DeathYear>1519
  </DeathYear></DeathDate>. <Nationality>Italian</Nationality>
    <Profession> painter, sculptor,
    architect, designer, theorist, engineer and scientist</Profession>
  </MasterData>.
  <Description>He was the founding father of what is called the High
  ...
  </Description>
</Artist>
```

Fig. 3. The result of a VaLa description

4.2 Iterative Development of VaLa Descriptions

The production of a VaLa description for automatic markup of documents, which explicit only marginal structure, is a difficult task. On the one hand, given large amounts of source documents, it is not possible to take into consideration every single document for production of the rules. On the other hand as many cases as possible should be covered by the rules of a VaLa description.

In order to achieve this goal we provide for the *Vasari* tool for interactive and iterative development of such descriptions. Similar as with MarkItUp! [10] VaLa descriptions are developed from example source documents. The knowledge engineer starts with the definition of the structure it is aimed at for the

resulting XML documents. The filler rules can then be developed by means of the example documents. At any stage of the development process the knowledge engineer can check the result by means of the example documents.

A VaLa description obtained in this way now can be improved iteratively. In each iteration step the knowledge engineer gives feedback to the system with regard to the result obtained up to then: The markup can be assessed as being wrong or correct; missing markup can be introduced into the XML documents. According to this kind of feedback, the VaLa description can then be improved further. Whenever a new version of the description is finished, the example documents are marked up using that version. Since feedback is available from earlier versions already, part of the assessment of the new result can be done by the system automatically and visualised to the user.

Figure 4 shows the main window of the Vasari user interface. The bottom part of the window contains the VaLa description developed up to then (see also Figure 2). The remaining upper part of the window is split into three parts: The left-hand part contains an overview of the example source documents. One of the source documents is displayed in the middle part, while the result of the markup process (using the VaLa description shown in the bottom part of the window) is displayed on the right hand side. As can be seen the artist's name and birth place has been marked up correctly. The knowledge engineer therefore marked the tags accordingly. Whenever markup of this part of the document is changed by any later version of the VaLa description, it is marked as being wrong automatically.

4.3 A Toolkit for Automatic Markup of Plain Texts

When developing means for markup of rather unstructured plain text documents, obtained from OCRed texts, we realized that there are some problems which are not directly related to automatic markup. This includes the correction of systematic OCR errors, the detection of document boundaries and the elimination of hyphenation of words in the pre-processing phase for the source documents. Also, some features of the VaLa language required the use of external tools, like SPPC to detect linguistic categories or entities. In order not to burden Vasari and VaLaEx with these tasks we developed a toolkit framework, of which VaLaEx is the core. Other tools for specific tasks in the markup process can be added arbitrarily.

All tools in the toolkit comply with a simple standard interface: The input as well as the output always consist of (sets of) XML documents - on the input side additional parameters, e.g. a VaLa description for the VaLaEx tool, might be provided. Hence a high degree of modularisation is achieved, and tools can be combined in an almost arbitrary way. The interface standard allows for easy integration of external tools, e.g. SPPC, by means of wrappers.

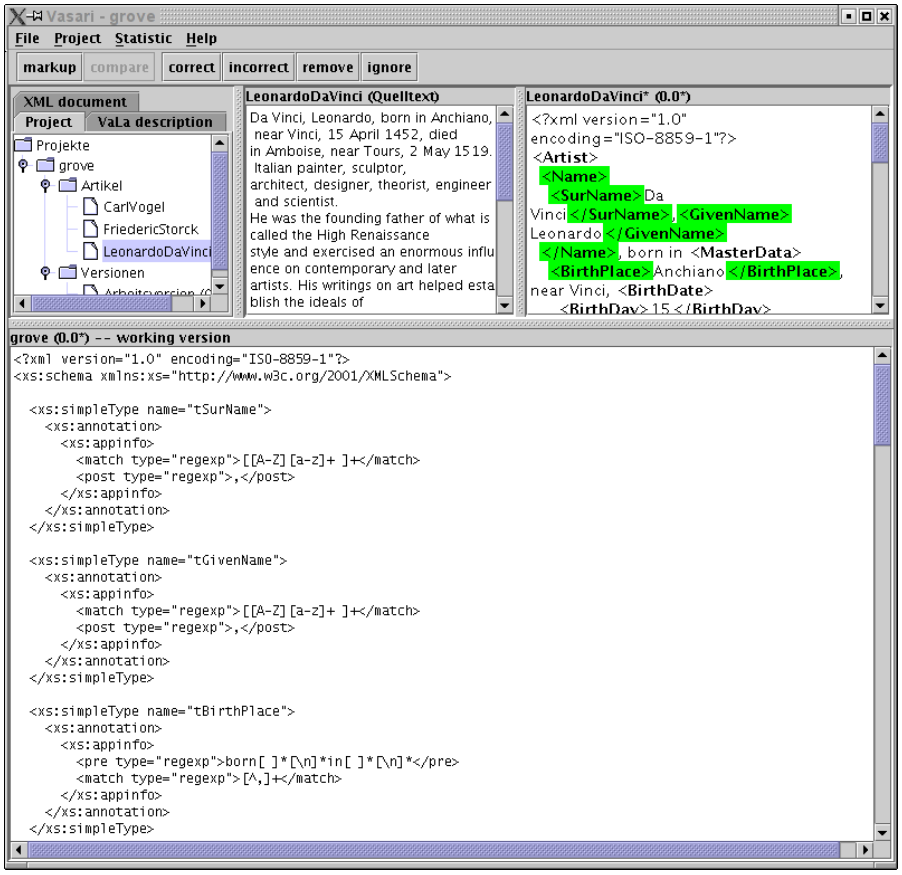

Fig. 4. Vasari user interface for interactive and iterative development of VaLa descriptions

4.4 Improving the results

Characteristic for the Vasari application is that source documents from different encyclopedias are available, the contents of which intersect partly. This can be exploited to achieve an even better knowledge representation after the automatic markup process is completed. Knowledge from different sources can be fused, thus references implicitly available within the source documents can be made explicit. For example, given that an artist is described within different encyclopedias, the fused descriptions would lead to a more complete view on that artist. Even contradictions could be detected, triggering e.g. manual correction.

5 Conclusion

Automatic markup of (legacy) documents will remain a problem for a long time. Using XML as the target format for automatic markup leads to powerful search and navigational structures for effective knowledge exploration. In this chapter we summarised approaches for automatic markup of macro and micro structures within rather unstructured documents. In a case study we demonstrated how automatic markup is applied to build a Web portal for arts.

Future research in this area will focus on the development of markup and extraction methods which are less domain-specific.

References

[1] Kaszkiel, M., Zobel, J.: Passage Retrieval Revisited. In Belkin, N.J., Narasimhalu, A.D., Willet, P., eds.: Proceedings of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, New York, ACM (1997) 178–185

[2] Southall, R.: Visual structure and transmission of meaning, Cambridge University Press (1988)

[3] Taghva, K., Condit, A., Borsack, J.: An evaluation of an automatic markup system. In: Proceedings of the IS&T/SPIE 1995 International Symposium on Electronic Imaging Science and Technology. (1995) 317–327

[4] Hitz, O., Robadey, L., Ingold, R.: Analysis of Synthetic Document Images. In: Fifth International Conference on Document Analysis and Recognition. (1999)

[5] NIST: Proceedings of the 7th Message Understanding Conference (MUC), NIST (1998) `http://www.itl.nist.gov/iad/894.02/related_projects/muc/proceedings/muc%_7_toc.html`.

[6] Appelt, E., Israel, D.: Introduction to Information Extraction Technology. A Tutorial Prepared for IJCAI 1999 (1999) `http://www.ai.mit.edu/people/jimmylin/papers/intro-to-ie.pdf`.

[7] Grishman, R.: Information Extraction: Techniques and Challenges. In: Information Extraction: A Multidisciplinary Approach to an Emerging Information Technology - International Summer School. Volume 1299 of Lecture Notes in Computer Science., Springer (1997)

[8] Cunningham, H.: Information extraction: a user guide (revised version). Prep, University of Sheffield (1999)

[9] Crespo, A., Jannink, J., Neuhold, E., Rys, M., Studer, R.: A survey of semi-automatic extraction and transformation. Technical report, Stanford University (1994)

[10] Fankhauser, P., Xu, Y.: Markitup! An incremental approach to document structure recognition. Electronic Publishing **6** (1993) 447–456

[11] Califf, M.E.: Relational Learning Techniques for Natural Language Extraction. PhD thesis, University of Texas at Austin (1998)
[12] Soderland, S.: Learning Information Extraction Rules for Semi-Structured and Free Text. Machine Learning (1999)
[13] Freitag, D.: Machine Learning for Information Extraction in Informal Domains. PhD thesis, Carnegie Mellon University (1998)
[14] Ciravegna, F.: Learning to tag information extraction from text. In: workshop on Machine Learning for Information Extraction. (2000)
[15] Vollmer, H., ed.: Allgemeines Lexikon der bildenden Künstler des 20. Jahrhunderts. Deutscher Taschenbuch Verlag, München (1992) 6 Bände; Nachdruck; Original von 1962.
[16] Fallside, D.C.: XML Schema Part 0: Primer. W3C recommendation, World Wide Web Consortium (2001) `http://www.w3.org/TR/xmlschema-0/`.
[17] Thompson, H.S., Beech, D., Maloney, M., Mendelsohn, N.: XML Schema Part 1: Structures. W3C recommendation, World Wide Web Consortium (2001) `http://www.w3.org/TR/xmlschema-1/`.
[18] Biron, P.V., Malhotra, A.: XML Schema Part 2: Datatypes. W3C recommendation, World Wide Web Consortium (2001) `http://www.w3.org/TR/xmlschema-2/`.
[19] Neumann, G., Piskorski, J.: A Shallow Text Processing Core Engine. Journal of Computational Intelligence **18** (2002)
[20] Lassila, O., Swick, R.R.: Resource Description Framework (RDF) Model and Syntax Specification. W3C recommendation, World Wide Web Consortium (1999) `http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/`.

Glossary

information extraction
: Information extraction is the process of automatically extracting specific interesting information with respect to a particular domain, including entities, relationships and events, from (the relevant text) documents, based on predefined templates.

automatic markup
: Automatic markup is the process of marking or tagging a document in order to specify and indicate its global visual and logical structure (macro level) and/or single words or word groups (micro level).