Pergamon

Information Systems Vol. , No. , pp. 1–, 1994
Copyright© 1994 Elsevier Science Ltd
Printed in Great Britain. All rights reserved
0306-4379/94 $7.00 + 0.00

A SURVEY OF SEMI-AUTOMATIC EXTRACTION AND TRANSFORMATION

Arturo Crespo[1], Jan Jannink[1], Erich Neuhold[2], Michael Rys[3] and Rudi Studer[4]

[1]Stanford University, Stanford, CA 94305, USA
[2]Darmstadt U. of Technology, GMD-IPSI, D-64293 Darmstadt, Germany
[3]Microsoft Corp., One Microsoft Way, Redmond, WA 98052
[4]University of Karlsruhe, Institute AIFB, D-76128 Karlsruhe, Germany

Abstract — This paper studies the *extraction and transformation* problem on documents. Solving this problem entails extracting the structures contained in a document and transforming the structures to make them available for further automatic processing. This paper provides an overview of methods and tools for extracting information and transforming the extracted information as required by the end-user. The overview is based on a taxonomy that classifies both characteristics of the available sources and properties of the extraction techniques. The paper concludes with a perspective on future developments, including a discussion of tools with learning capabilities and the role that XML and other related standards will play in the future.

1. INTRODUCTION

The unprecedented growth in the amount of information available in electronic form has brought upon us the understanding that our capacity to read, select, evaluate and synthesize information is quite limited. One way of expanding this limit is to make use of tools that can extract and transform electronic information. For instance, search engines and shopping bots are services that help us find the pages and products we want among the millions of pages and thousands of vendors on the World Wide Web.

The main challenges in creating such Extraction and Transformation tools are: (i) Information is offered in a variety of forms and expressed in a variety of contextual frameworks. (ii) Differences in structure, correctness and regularity make understanding and interpreting the information difficult.

We can readily find examples of these difficulties on the Web. Specifically, despite the fact that personal home pages usually contain quite similar information, the individual tastes in organizing and structuring that information vary widely. Likewise, the informational pages of companies' web sites contain very similar types of information structured each in their own way. Automatic processing is especially useful when dealing with very large amounts of information that makes systematic human reading and interpretation impractical, tedious, and error prone.

Our model for tools that aid in the *extraction and transformation* (henceforth *E&T*) task obeys the following definitions. An *extraction process* accesses data from one or more sources, and applies *extraction rules* to the source data as the tool accesses it. The set of these rules form an *extraction description*. Some descriptions are static, others change through a *rule generation* process initiated in response to source changes. To export consistent data and schema information to the end-user, the tools apply *transformation rules* based on the application of the end-user.

In summary, the goal of information processing tools defined above is to provide methods for extracting the hidden structures from a source, identifying the information contained in those structures, selecting the information relevant for a specific purpose, and transforming the selected information into coherent output structures that can easily be processed further by humans as well as computers.

However, we should not forget that information is ultimately intended for the human consumer. Computer processing, at best, represents an intermediate step, transforming, filtering, abstracting and presenting information required at a specific point of time and place by the human information seeker. Search engines on the web are early and, for the purpose of helping to locate information, sometimes quite successful examples of automatically extracting information from the vastness of the World Wide Web and offering it in a unified format to the human information seeker.

Their information processing power however is very limited as they are incapable of restructuring, filtering, or processing the contents of the information elements such as web pages, parts of pages or multi-page documents they identify.

Three principal approaches emerge from the body of work of the last few years to extract and transform information available on the Internet and other distributed systems:

Explicit Schema Many sources describe the structure of their contents quite explicitly with a database schema. Examples of this are traditional databases, both relational and object oriented, and even some more specialized formats, such as literature references (e.g., the Chemical Abstracts: CA, CAPLUS, CASREACT, REGISTRY.) An explicit schema appears to make the solution to the problem of extracting information trivial. In reality, the problem is still complex because of two factors. First, different databases may use different schema languages for these descriptions. Second, even when using the same languages, many structurally different ways of expressing the same information are possible. The problems of working with multiple databases and schemas have been identified in the literature by names such as interoperability [34], heterogeneous schema integration [19], global schema [31], and view integration [6]. All of these methodologies attempt to provide for the application and the users a unified and therefore simplified interface that allows for easier human interpretation and easier automatic processing of a multiplicity of databases. This field has been extensively explored in the literature, since this research deals with, in a sense, fully structured data. In the remainder of the paper, we will not further deal with those issues besides giving a brief overview in Section 2.

Information Retrieval Digitized textual and multimedia information usually does not have an explicit well-defined structure. Sections, paragraphs, indentations, lists, tables, as well as figures, graphs, images, and even dynamic components such as audio, video, simulations are described by markings for layout and presentation purposes. Examples of this formatting include MS Word, HTML, etc. These layout markings do not describe or even characterize what is contained in the documents, but are exploited in various extraction systems [16]. Information retrieval, computer linguistics and, more recently, image, graphics, audio and video analysis techniques, have been used to locate documents in a large set and interpret the contents of such documents. Approximate reasoning techniques for making the most relevant information available for the information seeker are frequently involved. Web search engines [14] serve as basic instances of such tools. The many IR publications, such as, [18] and IR systems present more advanced approaches, frequently employing context information (knowledge bases, thesauri or alternatively grammatical principles of natural language), as well as statistical exploration to better recognize relevant information and constrain the problem search space. Deeper analysis is achieved by computer linguistic approaches. In restricted domains, those approaches have been quite successful, as in the information technology domain [24] or in the message understanding approach for news articles on terrorist incidents [30]. However, in broader domains, scalability and functionality are still missing. Muslea [40] provides a thorough overview of such work as applied to the web and networked information. Because the computer linguistic approaches have been analyzed and compared in the past, we will not further dive into this field besides giving a brief overview in Section 2.

Implicit Structures In many types of documents, a significant amount of structural information actually exists, but is only given implicitly. In this case, a human expert of the domain is readily able to identify those structures. Examples of this implicit structure are a letter, a literature reference, a resume, and the description of works of art. These structures are readily recognized despite the fact that many differing versions exist. For computer analysis those structures, however, are not sufficiently identified. From this situation, the wide research and development field of structural extraction recently emerged. Such information is variously called unstructured, semistructured, partially structured, or probably most correctly implicitly structured [1]. Approaches to make these structures explicit and make them available for human and, most importantly, computer processing vary widely. Successes are very difficult to judge due to widely varying terminologies, taxonomies and metrics. However, some of the results of such structural extraction processes have shown themselves quite successful and scalable to large applications [13]. In the main body of this paper, we will provide an analysis of the problems encountered and the solutions proposed or respectively still to be found for the E&T problem. We will present a taxonomy for the different aspects of abstraction, describe its properties and already existing related work and results. However, due to the explosive growth of research in this area, it is impossible to achieve completeness in our literature references.

It is worthwhile to observe that developments like the advent of SGML [21] and the rapid acceptance of XML [22] offer hope that in the future some of the information will be generated with more explicit structuring information and that the E&T task can be simplified to just transformation. However, as the database schema interoperation field shows, even explicit structuring leaves a lot of difficult problems to be solved due to multiple ways of structuring a schema and content and the improbability of agreement on a single universal schema.

Solutions to the E&T problem rely on the fact that in many cases documents do not appear individually but that a large number of only slightly varying documents exist. As mentioned before, letters, references, descriptions, even things like journal articles or newspaper clippings have similar but slightly varying structures. Slowly developing strategies to represent "the right" corporate image on all web pages of an organization also lead to groups of similarly structured documents. Two approaches to E&T are distinguished and will be elaborated upon in more detail in the next paragraphs.

The first approach is to have a human domain expert develop the necessary extraction, transformation and filtering algorithms for each of those classes of documents. Although this process leads to high quality solutions, it has been shown to be labor intensive, error prone, time consuming and frequently prohibitively expensive. These difficulties arise especially in attempts to reuse earlier specifications when slight variations in the document structures are encountered. These variations arise through errors, newly encountered variations in the information contents, or requirement changes over time.

The second approach starts with an initial structural and transformational description together with learning mechanisms in order to develop systems capable of dealing with both slightly varying sets of documents and document structure evolution over time. Users of such systems accept that the learned structures are probably not always the most efficient and that complex situations may not be handled. For this reason, even powerful and scalable mechanisms will require human intervention. These *semi-automatic* learning techniques appear to be the best approach to relieve some of the burden from humans.

This paper is organized in four main parts. First, in Section 2, we briefly describe the Schema Integration and Linguistic approaches. Then, in Section 3, we present a taxonomy of source properties with relation to the extraction description generation. In Section 4, we classify the generation of extraction descriptions. Finally, in Section 5, we present interesting directions for future work on the E&T problem.

2. FIRST APPROACHES TO THE E&T PROBLEM

2.1. Schema Integration and Interoperation

Syntactic and semantic transformations have been investigated in depth for database integration. This involves mainly two kinds of transformations: data model translation [20] and view/schema integration [5]. Both types of transformation assume that there exists a complete, unambiguous and regular schema comprising the static model as well as the integrity constraints. Data model translation is concerned with generating transformations from a source data model into a target model in such a way that the semantics of the original model are preserved. Examples of data model translation are the generation of relational schemas with dependencies from an entity-relationship design [43], the generation of semantically more explicit schemas from relational schemas [8, 60], translations between object-oriented and entity-relationship models [20], and more generally between abstract models with partially ordered modeling capabilities [35]. The goal of view and schema integration is the generation of transformation rules that map two or more autonomously-designed database schemas having the same data model into a uniform integrated view. In this process, redundant schema portions are detected and removed, and conflicts are resolved in such a way that the integrated schema can represent all the original information without loss. Early work on the relational model [6] focused on resolving conflicts among different database schemas arising from imposing inter-schema constraints on top of intra-schema constraints, and developed means for integrating heterogeneous attributes [9]. Recent work on integrating object-oriented schemas has analyzed the formal properties of integrated schemas with respect to information preservation [39] and extended the transformational expressiveness of rules to cope with more kinds of modeling differences [53, 52, 31, 9].

2.2. Linguistic Analysis

There exists a long research tradition in computer linguistics for the problem of extracting the meaning from free texts. Typical linguistic approaches rely on a syntactical and semantical analysis of the given text source. In the context of information extraction, the problem to be tackled is much simpler than the full linguistic extraction problem for two reasons. First, those parts of the text that are not relevant to the end user may be ignored. Second, the information to be extracted is typically simple and has some structure [40].

Linguistic-based approaches to the E&T problem exploit to a varying degree lexical, syntactical and semantic information when parsing the source text. Since there are many different systems being described in the literature, we will not aim for a comprehensive overview, but rather sketch some main approaches that have been developed in recent years.

To describe the text parts to be extracted from the source text, the AutoSlog system [47] uses a combination of syntactical patterns, such as, `<subject> passive-verb`, and semantic information, like `<subject> physical-target`. In the former case, AutoSlog requires that the verb be in passive form, in the latter, that the subject be a physical target.

Within the PALKA system [30], meaning frames and phrasal patterns are exploited for guiding the extraction process. Meaning frames specify semantic constraints for the information to be extracted, like `agent: ANIMATE`, whereas phrasal patterns define the syntactical structure of the matched sentence, as in, `(target) was bombed by (agent)`. The first constraint indicates that the agent has be of type ANIMATE, and the second indicates that the sentence has to be in passive mode. Essentially, the PALKA system uses constructs that are similar to AutoSlog.

A more general approach is provided by the CRYSTAL system [51]. CRYSTAL offers means for specifying lexical, syntactical and semantical constraints for all parts of a sentence. Specifically, constraints can be established for the subject of a sentence, e.g., a lexical constraint like `terms include: BUILDING`, or for a prepositional phrase, e.g., a semantic constraint like `classes include: Person_Name`. In the above examples, CRYSTAL constrains the subject to include the word 'building' and forces the prepositional phrase to contain a person name.

In essence, the linguistic based approaches rely on some notion of patterns that specify lexical, syntactical and semantic constraints for the information to be extracted. Of course, the more semantic constraints are specified, the more the prior modeling of the semantics of the application domain becomes necessary. The accuracy of the semantics must also be verified during the extraction process. This clearly indicates that there is an obvious tradeoff between a more precise result of the extraction process and the complexity of the extraction process itself.

3. SOURCE PROPERTIES

As a first step in comprehending the E&T problem, it is important to understand the properties of the information source. The source properties will have a strong influence on the complexity of the E&T problem as well as on the space of possible transformations that we can apply to the information. The most significant properties are presented in form of a taxonomy in Figure 1 and are discussed below.

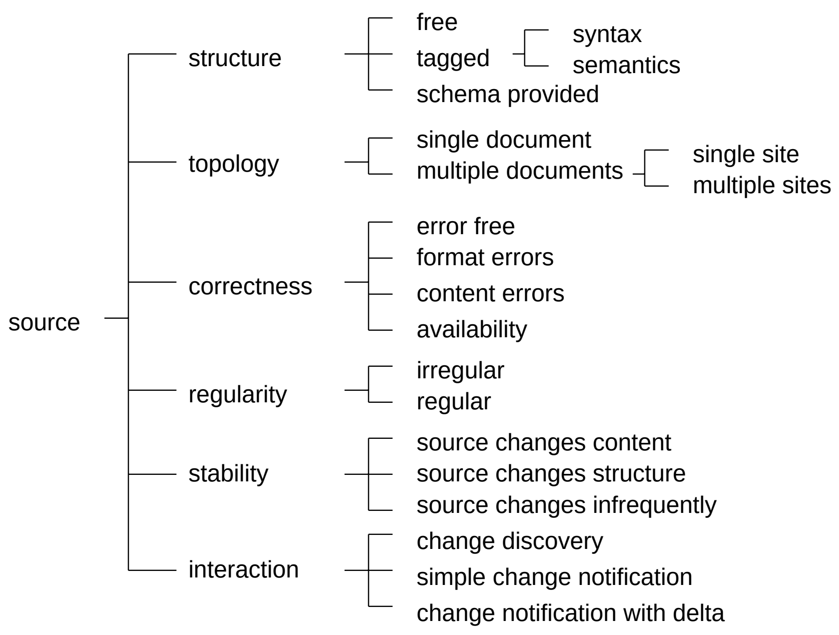

Fig. 1: Taxonomy of source properties relevant to E&T

3.1. Structure

The structure of the source is an important factor in deciding how to generate extraction descriptions. Structure here refers to syntactic marks and formatting that organize the content of a document. Sources can vary from being completely unstructured, i.e., free text, to having some implicit structure, e.g., section numbers in a text document, to being fully structured, such as a relational database.

Today, many sources structure their data by using tags. For example, most web documents are marked up with HTML, which explicitly provides the document structure in a syntactical way. Other tagged documents are SGML and XML documents, where the tagging conveys some semantical information about the source data and leaves the syntactical aspects to the associated style files.

In other instances, the sources may provide meta-information about the document such as a schema. For example, an XML source might provide the schema in form of a document type definition (DTD). In addition, ontologies may be used to offer semantic meta-information about a source document [26, 12, 15]. The influence of the source structure on the extraction process will be discussed in detail in Section 4.4. When the document provides little structural information that can be used during extraction, the following properties still affect the automation of the extraction process.

3.2. Topology

The topology of a source refers to the different documents at a source and the relationships between each other. In order to fully understand the implication of topology on the extraction description generation, it is necessary to clearly define what a document is. The difficulties associated with defining a document can be illustrated with the case of web sites. A web document, from a purely syntactical standpoint, is a single HTML file, together with its embedded, uninterpreted components (GIF, JPEG, applet.) However, multiple HTML pages often form a semantic unit through links between each other. Many applications, such as latex2html and javadoc, convert a single original document into a set of web pages. Many handcrafted web sites likewise are semantically a unit, despite consisting of many pages. Therefore, a web document may consist of many HTML pages, even spanning multiple sites In contrast, a full text document, such as a book, forms a single physical document. However, for extraction purposes, there are applications where it is reasonable to consider the physical document as a sequence of documents, each of which describes a chapter.

The notion of document has no single mathematical definition. In general, recognizing larger structures as single entities must proceed from a semantic basis. Therefore, the actual definition of a document must be provided by the end user of the data. In fact, semi-automatic extraction techniques, discussed in Section 4.1.2, typically depend on manual specification of document boundaries. In addition, Section 4.5.2 discusses aspects of defining document structure based on the target document semantics.

Let us turn our attention from the topology of a single document to the topology of a set of documents. When documents are interconnected, as in the example of chapters of a book, their interrelationships are valuable and the extraction process must recognize and preserve them.

3.3. Correctness

A major issue when trying to automate the generation of extraction descriptions is the anticipated correctness of the source data. Errors can range from simple misspellings, omissions and superfluous additions, structural errors, to completely erroneously specified documents. Later, in Section 4.3, we will present a taxonomy of techniques to deal with the correctness of source data.

The source can be considered error free or having format, content, or availability errors (or combinations of them). Format errors happen when the page does not conform to the expected format (e.g., an XML page that do not conform to its DTD). As another example of format error, consider the case of a bibliography entry that does not conform to the expected Springer citation rules but instead follows the ACM Press rules. Content errors happen when the page does not contain the information that we were expecting. For example, an XML page that we expect to contain prices of items, but instead contains quantities in inventory. A different category of errors derives from source availability. For example, the source may be stored remotely and the extraction engine has to transfer the data locally. Assuming that the transmission layer is handling erroneous transmissions, the main problem is the possibility of a temporal unavailability of the source, due to a problem with the source server or due to network partitioning. Section 4.3 discusses error handling related to sources. Of course, the source may provide a guarantee that it is free of errors or at least free of certain kinds of errors. In this case, the extraction process can be much simpler than if the source does not make any guarantees.

3.4. Regularity

Instances of source documents may be regular in their format or vary widely. The latter case usually occurs when the document instances are hand-coded or generated over time. Irregular documents are not a priori incorrect but pose similar problems to the extraction process. The contents of document instances may have developed over time and may contain components that were not previously allowed or are not allowed anymore. Change over time introduces irregularities to the source that have to be taken into account by the extraction description generation.

For example, the document instance being analyzed may contain component variations not encountered before and therefore not included in the extraction description. In practice, lack of data regularity is a problem so frequently encountered that it should be considered and treated as unavoidable. Section 4.2 covers the computing power of the extraction process, and its impact on handling source irregularity.

3.5. Stability

Besides the evolutionary change of the document instances, which leads to source irregularities, sources can also change more abruptly by changing their format or content at once. While it is unusual that a source changes its topic or *domain*, it may well often change its domain specific content. For example, a news service replaces old news articles with new ones. Likewise, stock quoting services constantly update pages based on current market prices.

Especially in commercial information sources, changes to the format can happen frequently due to corporate identity changes, advances in technology (i.e., the introduction of JavaScript or XML), or "simple" presentation design changes. Unlike the case of irregularities introduced over time, these changes normally affect all document instances at once. Therefore, the extraction process does not need to keep track of the old versions but still needs to be able to detect the changes and regenerate the new extraction descriptions.

3.6. Interaction

Some sources are able to provide notifications whenever its data changes. This notification can be useful for the extraction process, as frequent changes will invalidate an existing extraction description. Sometimes the notification includes what is changing, the *delta*, but more often the change notification only informs of a change and does not provide the delta. In the latter case, the extraction process needs to discover the changes itself, in order to bring itself up to date. Otherwise, if the change notification provides the change delta, this information can be utilized directly to regenerate the new extraction description. If no change notification is provided by the source, the extraction process may or may not be able to detect the change and in the worst case may return invalid data. See the discussion of change handling by the extraction process in Section 4.3 below.

4. EXTRACTION PROPERTIES

In this section, we explore the different properties of the extraction process. Although the extraction process is partially determined by the properties of the target source, there is still some flexibility in establishing the extraction process. For example, when a source contains an error (a source property), the extraction can choose to fail, can attempt to recognize the error and generate a warning, or can modify the E&T rules to account for the error. In Figure 2, we give an overview of the properties that we will study in this section.

4.1. Automation

An important property of the extraction process is its level of automation. An extraction process can be ad-hoc, based on rules specified manually with or without tool assistance, or even automatic.

When a programmer is first confronted with the need of solving an E&T problem, the common reaction is to write some ad-hoc code to solve the problem. This solution is acceptable when there are a small number of stable sources. However, as the number of sources grows (or they become less stable), the programmer finds himself writing very similar (although not equivalent) pieces of code to perform the extraction. Manual approaches are based on the idea of capturing this similarity in extraction rules. Specifically, this is achieved by providing the programmer with libraries and tools such as W4F [48] that generate most of the extraction code based on extraction rules defined by the programmer. Unfortunately, writing extraction rules is labor intensive and error prone. For example, when refactoring the structure of an on-line version of the Oxford English Dictionary (OED) [44], it was found that fully 7% of the dictionary's original on-line definitions were erroneous and contained only punctuation or single words like 'hence' and 'also'. The next level of automation, offered by semi-automatic extraction systems, is to enhance the process of generating extraction rules. This automation is typically achieved with a learning-based system, where the programmer directs the process until it achieves the required quality. Semi-automatic systems frequently offer the best tradeoff between quality of translation and resources used. Finally, automatic systems attempt to replace completely the role of the programmer with an Artificial Intelligence system.

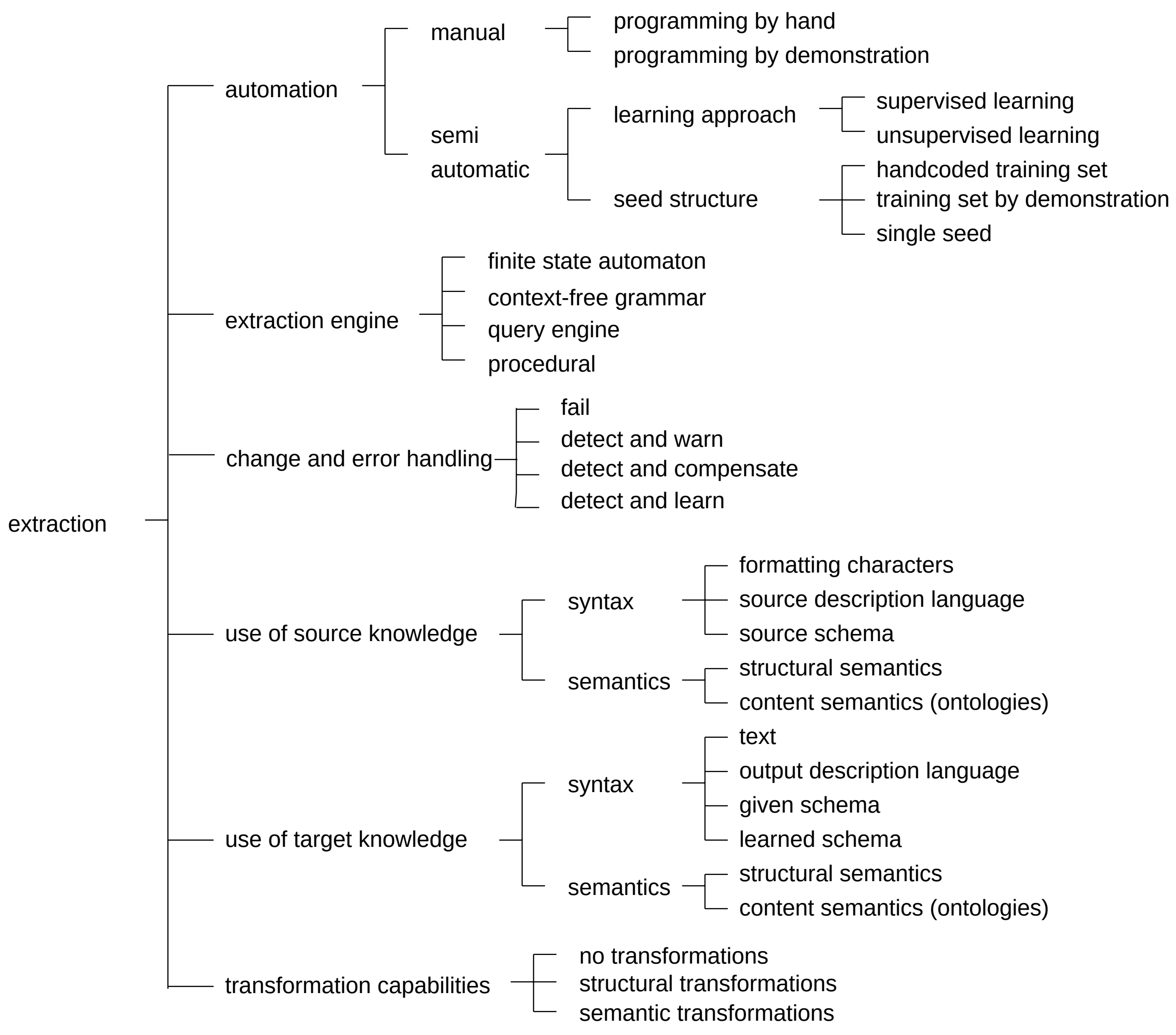

Fig. 2: Taxonomy of extraction properties relevant to E&T

In the following subsections, we will further discuss the manual and semi-automatic approaches to the generation of extraction rules.

4.1.1. Manual Extraction

We define a manual approach to the E&T problem as one where users are required to describe the extraction process by writing rules in some high level language. Specifically, when using a manual approach, the problem is solved by having an expert user understand the syntax and semantics of the source and write the procedure to translate the source into the target data structure.

A designer of a system based on a manual specification of rules is confronted with two, sometimes conflicting, objectives. First, the system has to be powerful enough to handle the extraction process. This is, given a source and a target structure, we should be able to instruct the system to extract the information from the source and arrange it in the style of the target structure. Second, the system should also allow for easy specification of the extraction rules. In other words, the system should be easy enough, so it is worth using it instead of writing an extraction program in a traditional programming language.

We now present two examples from current research. The first example, Proxygen [25], focused in the first objective, ease of use, while the second example, EDITOR [4], focused on obtaining a language that is computationally complete. Both systems require an expert user to write extraction rules in a *template* or *script.*

Proxygen provides the user with a toolkit that contains many constructs, specially for HTML processing, that makes writing an extractor easier that when using a programming language or a tool like LEX or YACC. The user specifies the processing in a procedural way, with statements that are processed sequentially. Statements have three components: a source variable, which contains a list of strings; an operation, which can be a generalized regular expression or a library function call; and, a target variable. The execution of a statement is simply applying the operation to the source variable and placing the result into the target variable. When the operation is a regular expression, the user can specify a pattern to find as well as what to extract from that pattern. For example, if the operation is the regular expression `*author: #.`, and the source variable is the list `["paper author: XXX.", "paper author: YYY"]`, the target variable will be assigned the value `["XXX","YYY"]`. In the pattern, the '`*`' means any sequence of zero or more characters; the '`#`' saves the resulting pattern. Proxygen also provides error-handling operations. Each statement can have a `case` instruction, allowing the system to try different regular expressions (or functions) until it finds one that matches (or returns failure).

The system in [4], EDITOR, takes a different approach. Instead of providing a large number of constructs like in Proxygen, it only has four source operations: *search, loop search, cut, and copy.* The *search* and *loop search* instructions are used to select regions of interest, while the *cut* and *copy* instructions are used to restructure those regions. EDITOR has been proved to be computationally complete, in the sense that any computable document restructuring can be expressed in it. For example, in Figure 3 the author list on the left is transformed into the table on the right by the script in the center.

```
paper          loop search(source, "paper author: *")      XXX
author: XXX       copy(source)                              YYY
paper             paste(destination)
author: YYY    end loop.
```

Fig. 3: Author List, Script and Output Table

The primary advantage of a system based on a manual generation of rules is that the extractors it derives are typically very efficient and fast. These characteristics allow this kind of extractor to process large volumes of data. Additionally, the complete control over the extraction process allows the programmer to handle complicated semantics that might be very hard to define by using a semi-automatic or automatic system.

The drawback of the manual approach is that the extraction mechanism depends on human expertise for describing the structure of the source documents. Hand-coding the pattern matching rules can become a complex, labor intensive, time consuming and error prone task. This drawback can be diminished by adding error handling capabilities to the extractor, which allows it to survive transient errors in the source and to exit gracefully when the source has changed. In addition, providing a GUI front end can make writing the extraction specification easier. As the user interface becomes more sophisticated, the extraction process starts to become a semi-automatic task, rather than a manual one. In the next section, we will explore semi-automatic approaches to the extraction problem.

4.1.2. Semi-Automatic Extraction

To reduce the labor intensiveness and the time consumption of the manual generation of extraction rules, the E&T process can be assisted by tools that at least partially automate the extraction. In order for the automation to yield extraction descriptions that are close in quality to hand-coded descriptions, the generation process needs to provide for some kind of quality control, as in the form of expert feedback.

A basic learning-based algorithm is divided in three phases: a training phase, a processing phase, and a feedback phase. In the training phase, the algorithm assimilates a set of correct samples of extraction solutions. In the processing phase, the algorithm applies the patterns from the training samples to new inputs. Finally, in the feedback phase, the user provides some measure for the quality of the extraction process. This feedback allows the algorithm to improve its "learned" behavior. Under continued use in the E&T process, the processing and feedback phases are repeated. We consider that learning-based approaches achieve a good balance between quality of extraction and ease of use.

Several characteristics define the training phase. One characteristic is whether the training phase is incremental or whether it proceeds in a batch mode. That is, either the set of training examples may grow over time, or they must all be specified in advance. A second characteristic concerns the language bias, i.e., what kind of representational formalism is used for describing the training instances and the learned hypotheses. Finally, a third characteristic is the search bias, which specifies whether the learning algorithm is performing a top-down or a bottom-up search, i.e., whether it starts from the most general hypothesis and specializes it or whether it takes the specific training examples as a starting point and generalizes them.

SoftMealy [27] pursues a bottom-up learning approach and assumes that all training examples are available from the very beginning. SoftMealy uses finite-state transducers to represent the learned wrapper. The STALKER wrapper induction algorithm [41] may be characterized as a top-down learning approach that refines landmark automata as long as there exist some uncovered positive training examples. Again, STALKER assumes that all training examples are given in advance.

In the extraction process, positive training examples can be either a complete source document or a source fragment characterizing elements that should be extracted. In the same way, negative examples could describe either a complete source or a source fragment that should not be covered by the generated extraction description.

A different learning strategy starts with a small seed and then automatically tries to discover new rules that can be used to extract data. It is obvious that it is hard to make any guarantees about the quality of the generated extraction descriptions due to the unsupervised progress and addition of rules.

4.2. Extraction Engine

Now we turn our attention to the engine that processes the extraction rules. The simplest E&T engines are based on finite state automata where the transitions describe the extraction process. An example for this kind of transformation engine is the Proxygen extractor [25]. In Proxygen, each transition defines an extraction using regular expressions and each state represents both the incoming and outgoing data on which the regular expressions operate. The advantages of finite state automata are their simplicity and ease of use. The transformation specification (i.e., the specific extraction automaton) can easily be changed and adapted to new states and extraction rules. Therefore, an automata-based engine is well suited for being used in semi-automatic extraction. However, a major disadvantage of the finite state automaton however is the limited expressive power, since it can only describe regular languages, i.e., it cannot express more than primitive recursion. This means for example, that it can only handle nested structures where the nesting depth is known in advance. Additionally, the number of transitions may be extremely high if the source changes often.

Transformation engines based on context free grammars can employ recursion to express extraction rules that can handle arbitrarily nested structure. Specifically, the grammar rules can be used to describe the expected input format while the annotations transform the parsed input and generate the extraction result. This category contains simple YACC based transformation engines that cannot deal well with changing or erroneous data sources, or engines based on fault tolerant parsers such as the one used by JEDI [28]. JEDI's parser, for example, can cope with incomplete and ambiguous source specifications by choosing the most specific rule among several applicable rules. When finding no applicable rule, it skips as little as necessary of the source data, until it finds an applicable rule. While creating context free grammars is, in general, more complex than generating finite state automata, these extraction descriptions are still well suited for semi-automatic generation.

The main advantage of pattern based finite state automata is that they can easily describe only the parts that are interesting for the extraction process and disregard everything else. On the other hand, standard parsing based grammars normally need to give an exact description of the document and are therefore less flexible. However, less brittle approaches such as the one employed by JEDI, make the grammar-based approaches more appealing. In addition, context free grammars have a higher expressive power than finite state automata.

Increasingly, the source data is already represented in a form that can be interpreted as a data model such as OEM [46], XML [58] or ASN.1 [33]. In this case, the transformation engines can be query engines supporting query languages such as Lorel [2] or XSLT [59] that can be used to describe the E&T process. The expressive power of these engines depends on the expressive power of the query languages. These query languages still need to be able to cope with the source properties from Section 3. Again, the generation of these views lends itself well to semi-automatic generation.

Finally, the most flexible, expressive category of extraction engines is based on procedural descriptions that, in the general case, are able to express context sensitive rules. However, writing these procedures is normally a very time consuming task where every single detail needs to be written out. Unless the procedural language is not limited, such as EDITOR [4], the semi-automatic generation is difficult and the resulting procedures require significant maintenance.

4.3. Change and Error Handling

Constant changes and the presence of errors are common characteristics of unstructured data. A good extraction system must be able to deal with errors in an effective way. This is particularly important in some applications (e.g., data cleansing) where the errors, rather than the data, are the focus of the extraction process. It is clear that a semi-automatic generation of the extraction description cannot predict all possible errors. However, the extraction mechanism can provide a spectrum of solutions, ranging from fault tolerance in the presence of errors to automatic modification of rules when abnormal data is found. We will first distinguish between errors and changes, and then we will outline the different alternatives for dealing with them during the E&T process.

We define an error to be an isolated abnormality of the data, which could be either spatial or temporal. A spatially isolated abnormality is a localized one that happens only in a handful of documents of the source (e.g., a typo in the document). A temporally isolated abnormality is a transient one that happens only for a short period of time (e.g., a document that is temporarily unavailable).

A change, on the other hand, is defined as a recurring abnormality of the data that has appeared since the extraction rules were originally written. For example, the contents of document instances may have developed over time and may now contain components that earlier were not allowed or that are not allowed anymore. This introduces an irregularity to the source that has to be taken into account by the extraction process. We call this irregularity a delta or a *change*.

Error detection and change handling mechanisms can be grouped in four major categories: (i) fail, (ii) detect and warn, (iii) detect and compensate, and (iv) detect and learn. Detected but ignored errors are not considered, because they are indistinguishable from undetected errors.

A *fail* strategy simply aborts the extraction process when abnormal data is found. This is the simplest approach to the problem and it is unacceptable in most situations.

A *detect and warn* strategy is able to find the abnormal data and can warn the user of its presence. Then, the user needs to correct the abnormal data, and restart the process again.

A *detect and compensate* strategy, not only finds the abnormal data, but also tries to compensate for its presence. The simplest compensation is just to disregard the erroneous data and continue with the extraction process. More sophisticated compensations include: applying a different rule, or retrying the original rule several times until it succeeds (in case of a transient error). An additional way of compensation, that is extremely useful for change handling, is the use of different "versions" of the extraction rules. More specifically, when the extraction mechanism considers a document, it first detects to which version it belongs and then applies the correct version of the extraction rules.

A *detect and learn* strategy, not only compensates in the current extraction process, but it tries to regenerate the extraction description based on the newly discovered anomaly. This learning process may be done with the help of a specialist.

4.4. Use of Source Knowledge

The more structured the source is, the easier it is to automate the generation process by exploiting the syntactical and semantical information encoded by the structure. For example, wrappers for web based information sources are typical applications where at least some form of structure is available in form of syntactical tags (HTML) or tags with user defined structure and implied semantics (XML).

On the other hand, there are many applications where the goal is to generate more structured documents from sources that provide almost no structure. For example, one might want to mark free text documents with SGML tags for further processing in a text database [28].

In many extractions, it simplifies the rule set to utilize knowledge external to the source documents and meta knowledge about the documents themselves and the document contents. For example, a thesaurus could be used to specify generalization and specialization behavior in order to make it easier to deal with semantic granularity differences in the input structures.

4.4.1. Syntax

The simplest formatting to exploit in source documents are white space, carriage returns, null and EOF characters. Countless source documents also use punctuation and capitalization or positional regularity as an implicit form of structuring. For example, programming languages use special escapes for comments, and email and net news programs use special characters to cite inserted text from other messages. All mark-up languages rely on escape characters to indicate commands and their parameters. Extraction based on these characters relies on the fact that they have a customary and uniform semantics [40, 3].

Source description languages and mark-up languages support the tagging of portions of documents for specialized processing. Most of these languages have a fixed set of commands, and are targeted to a single application. A separate application such as a search engine [45] may emphasize portions of the source information by capitalizing on the fixed interpretation of the embedded tags. Supplementary information such as file extensions provide context to confirm the command interpretations. However, SGML and more recently XML are extensible, and designed for use in general application domains. Here separate documents may use the same tags with different intended meanings. In these cases, XML namespaces associate context to interpret tag names. Although these description languages are convenient, they are inefficient when source data is very regular.

Database schemas, object class definitions, RDF schemas[56], and XML schemas (such as DTDs or Microsoft's XDR [37]) provide specifications for document structures. Extraction based on these specifications depends on the complete regularity of the source data. Any non-conforming or erroneous data will not be properly accounted for in this extraction setting. However, XML schemas allow open content and thus they are somewhat forgiving on certain kinds of irregularities of open-content data.

4.4.2. Semantics

In the preceding paragraphs, we have seen how syntax based extraction on source documents depends on being able to assign a single semantics to the structures of interest. The semantics can be uniform by convention, by definition, or through standardization. In certain cases the semantic can be user defined. For example, DTDs express (to a certain degree) the explicit structure of a XML document. It is possible to extract information across document syntax types, by respecting their common semantics. For example, we may compile lists of document titles from HTML, email, file names, etc.

Beyond extraction based on the semantics of the document's structure, we may invoke the semantics of the data contained in the documents. For example, the SKC project [29] extracts a list of bodies of water by finding names adjacent to the set of words Bay, Sea, Ocean, Gulf, etc. This operation exploits the ontological properties of the source documents' contents. Such operations are understood to be context dependent, and are not universally applicable across all sources. Ontobroker [12] explicitly attaches ontological facts to source documents to guide which semantic operations are acceptable over them.

4.5. Use of Target Knowledge

The target output of the extraction process often determines the approach we use in interpreting the source data. In particular, we can limit the scope and complexity of the extraction, by focusing on the target result. Obtaining a list of country capitals from an on-line atlas, for instance, does not require any information about the countries' natural resources. Similarly, when the target output is plain text, we may disregard all formatting tags in a source.

General knowledge bases can support semantic mappings between input and output documents and reduce the number of alternative analysis paths to be considered. Examples are medical classifications like MED or MEDLINE [55] that simplify the task of extracting and completing drug description information for pharmaceutical data bases. Meta-information, such as the Dublin Core [11], reduces the effort required for developing the rule sets by reducing both the number and complexity of the E&T rules.

4.5.1. Syntax

When the target format is plain text, typical transformations remove tagging and explicit structures. Flattening a structured source to a textual format is the simplest use of target requirements in an extraction. When passing a document to a spell checker the flattening operation is very beneficial. It is often convenient to add white space and carriage returns to documents to simplify their further processing.

The translation of tags and implicit structure in documents to an output description language is common. XML is becoming a popular output format for semistructured data of all types. Since XML is a serial encoding, we may consider serializing source documents before considering further extraction. In addition, we must define the granularity of the encoding we will use to transform the source data.

Given a schema defining the target data, we may project out unimportant parts of a source schema during extraction. We also remove tags in the source documents that are defined by the target schema. If we are learning a target schema, we must drop irregular portions of the source documents. Likewise, we lose data for which there are not enough training samples to generate a schema.

An example of a transformation language based on syntax is XSL (Extensible Stylesheet Language) [59]. XSL consists of two parts: a language for transforming XML documents, and an XML vocabulary for specifying formatting semantics. Basically, An XSL stylesheet specifies the presentation of a class of XML documents by describing how an instance of the class is transformed into an XML document that uses the formatting vocabulary.

4.5.2. Semantics

With a defined semantics for the target information, it is possible to further limit how much of the source data is examined. For instance, in the case of textual output we may drop all portions of sources that are non-textual. When extracting structurally regular portions of a source into a database, it is simplest to filter the input using regular expressions. It is common to prune multi-file source documents based on the desired target structure. For example, the extraction of data on Mediterranean fisheries will be made more efficient by excluding source files relating to the Pacific and Indian Oceans.

An output ontology is a powerful tool to guide the extraction process. Ontologies provide support for determining validity of source data, for example. In addition, the output ontology can drive the filtering of source documents during the course of extraction. In the example of extracting country capitals, the output ontology allows us to filter all portions of the on-line atlas, aside from the political data [29].

4.6. Transformation Capabilities

Unless the extraction just copies the input data to the output as is, any extraction performs some kind of transformation. There exists a whole range of transformations that either could be part of the extraction process or could be handled after the extraction phase in a special transformation phase, for example, by view definitions over the extracted data. The best distribution of the transformations between these two phases has to be carefully decided based on criteria such as the data models and extraction engines involved in the process. In the case of semi-automatic extraction, transformations that can efficiently be handled automatically are best left to the later transformation phase.

In case we deal with documents where the input and output structure description languages are the same, such as Springer citation rules or SGML documents, we can make all implicit structures explicit by full SGML tagging or add missing components with either default or null values. This process refines the existing structure by adding new tags where the page content allows it. Necessary mechanisms here may well go beyond learning and may involve design decisions by humans. These design decisions should be moved to the later transformation phase, in order to keep the extraction process semi-automatic.

In addition, we can eliminate unwanted structural components by applying filters to the input documents. The filters focus the extraction process on application specific information. In this case, the rule specification process can be considerably automated by using a rule set to develop a structural description of the input and output documents. For instance, the process can generate a schema for XML documents that then can be used to generate the E&T specification.

If the structural description languages of the input and the output are different, the extraction rules will not only have to make implicit structures explicit, but will have to translate them into well formed structures of the output language. For example, the input format may be ICD-10[†], while the output format is SGML. The use of the embedded semantics of these languages is important and usually sufficient for these translations.

Sometimes the structure of the document instances has to be changed regardless of the input and output description languages, as when moving a subcomponent `name` in the structure `address` to the same level as `address`. In such cases, the semantics of the structural components themselves, –in our example above, the structures that are accepted as addresses– will have to be understood and utilized for the construction of the extraction description. Again, problem factorization and the variants discussed above will influence the complexity and power of the rule sets required.

Finally, the structural description of the output documents may not only require the semantics of the implicit structures of the input, but also an understanding of the contents of the input documents. For example, we must be able to extract the construction date but not the restoration date from building descriptions in tour guide documents. In this case, methods developed in computer linguistics may have to be used to extract such embedded information and place it into explicit structures of the output documents. Such *language understanding* issues are beyond the scope of our analysis.

[†]Tenth Revision of the International Statistical Classification of Diseases and Related Health Problems by the WHO

5. INTERESTING NEW DIRECTIONS

A number of promising directions for future work arise from the classification and analysis of the various approaches in the previous sections.

5.1. Training Examples

The training examples that are provided as input to the learning process characterize the subset of relevant information elements that should be extracted from an information source. All of the approaches discussed rely only on the specification of positive training examples. For example, NoDose [3] uses training examples to "learn" the proper way of decomposing a web page. First, the user shows a sample decomposition of a web page to NoDose. The program attempts to learn from that example and shows the result in a different web page. The user can then correct the program, which learns from the changes, and attempts to decompose another page. The process continues until all pages of the collection have been decomposed.

Negative examples are not considered at all in wrapper generation [27] or they are implicitly assumed by applying the closed-world assumption as in the STALKER system [41]. It is obvious that the explicit specification of carefully chosen negative examples could be exploited for guiding the learning process as well. Thus, the problem of over generalization could be reduced to some extent. Of course, one would need an appropriate user interface for specifying the negative training examples. Such an interface could be easily provided by generalizing the cut-and-paste interfaces that are available for specifying the positive training examples. The "demonstration-oriented user interface" of the Ariadne system [32] could be extended in a straightforward way for handling negative training examples as well.

In addition none of the techniques examined offer guidelines for choosing informative training examples, either positive or negative ones. It is up to the user to select most promising training instances. This situation is specific to the wrapper generation field since in other machine learning applications, the training examples are given from the very beginning and thus that problem does not arise. However, there exist active learning methods [10] for proposing additional informative training instances. The WHISK system [50] suggests three different types of new training instances: instances to increase the precision of rules, instances that can be covered by a slight generalization of an existing rule, and instances not yet covered by any rule. Further research is necessary to come up with good heuristics for proposing additional training instances.

5.2. Incremental Learning

A more interactive learning framework provides means for improving the learning process in various ways. For example, a tool based on an incremental learning algorithm could improve its extraction rules by using feedback from the user. This feedback does not necessarily have to be of positive nature. In fact, in case the wrapper extracts information items which do not meet the requirements of the user, these *false positives* items, could be offered as negative training examples. Thus, a stepwise refinement of the generated rules could be achieved.

An incremental learning approach helps to cope with evolving information sources that change their structure over time. By using an incremental learning approach, one could avoid to learn the wrapper rules always from scratch. Instead training instances from updated sources could be used as additional training instances for adapting the formerly learned rule set to its new environment, i.e., the updated sources.

5.3. Syntactic and Semantic Knowledge

Wrapper generation systems rely only on patterns or rules that exploit delimiters found in the source text, in contrast to approaches in the area of artificial intelligence such as in [7]. Wrapper generation systems make use of HTML tags and source text words which are domain specific, like 'high' or 'low' in weather forecast sources. Neither syntactical structures nor semantic constraints are needed to define the extraction patterns or rules. This is possible because these information sources exhibit typically a very simple and regular syntactical structure. This simplicity contrasts with the task of applications that extract information from free texts in the management succession domain [54].

Nevertheless, the usage of simple syntactical information, e.g., the usage of syntactic categories, could reduce the number of alternatives that have to be considered during the rule generation process. Furthermore, the usage of some kind of semantic knowledge, as typically found in a thesaurus or a system like WordNet [38] might improve the learning process in case the wrapper has to deal with irregular source structures or with varying terminology.

When analyzing sources like apartment rental ads one can easily see that a wide variety of words are used to describe the same real world object. Rental ads use terms like 'bdrm', 'brs' or 'bedrooms' to refer to the number of bedrooms of an offered house. Thus, a wrapper has to be able to handle homonyms, based on semantic relations as offered by WordNet. Job offerings often use terms on different levels of abstraction to refer to the same kind of job. If someone asks for a 'System Engineer' job, the person should get hold of job offerings for 'Software Engineer' or 'System Specialist'. In order to be able to deal with such situations, a wrapper generation system would need an ontology specifying semantic relationships like *generalization* between different job types. By using syntactic and semantic knowledge, wrapper generation systems become more similar to information extraction techniques. However, one has to be careful in adding this additional knowledge since wrappers might easily become too complex and thus too inefficient. Nevertheless, it is worthwhile to investigate some of these approaches, like SRV [17] or WHISK [50]. A nice overview of these kinds of systems can be found in Muslea [40].

5.4. Link Structures between Web Pages

A severe limitation of the discussed approaches is their restriction to the handling of a single web page. Many information sources are organized in a way that additional information is found by following links to further web pages. For example, when analyzing the source of *World Government* [23] important information about the structure of the government as well as its members may only be found by accessing a sequence of linked web pages.

Obviously, taking into account links to other web pages makes the learning process much more complicated since one has to learn in addition which links are relevant to follow and which are not. A combination of different features seems to be promising for evaluating the relevance of a link: (i) Some kind of typing can be used to classify links into different classes. Then, based on their class membership links may be determined as relevant or not relevant. (ii) An analysis of the structure of the target page might provide useful information to check the relevance of a link. (iii) The context in which a link is defined might also provide valuable information about its relevance. Of course, all these features need careful analysis in order to determine which features are most promising. An example of how to deal with this issues is in [36].

When considering future directions one also has to take into account the development of XML [58] and associated XML schemas (such as DTDs). The generation of XML wrappers clearly eliminates many low-level syntactical problems that currently have to be addressed. The problem of how to map a given source to a target schema structure in a semantically consistent way will nevertheless remain relevant, since XML does not provide that kind of semantic information. The development of the Resource Description Framework (RDF), see [57, 56], will have to be considered in that context, since that initiative aims at providing means for describing semantic meta data about a given web resource. In general, the development of XML and associated standards may shift the generation of a wrapper from a rather low-level syntactical problem to a more semantic problem.

6. CONCLUSION

In recent years, a number of approaches have been developed for extracting and transforming information from online sources. In this paper we introduced a two dimensional taxonomy for classifying the extraction and transformation task. The taxonomy's first dimension classifies information sources with respect to their structure, their correctness and their regularity. The taxonomy's second dimension classifies extraction methods according to features like automation, use of source and target knowledge, and transformation capabilities.

A widespread use of XML and related Document Type Definitions may solve some of the low level syntactic problems currently found when extracting information from heterogeneous sources. Nevertheless, XML tagging remains on a syntactic level, and lacks a clear semantic definition of the used tags. Because of XML's lack of semantic focus, further research is required on the relationship between DTDs, schemas, and namespaces on one side and ontologies on the other side. This relationship is important because ontologies may be used for providing a well-defined semantics for the XML tags. First steps are described in [15].

Another promising line of research is represented by approaches combining the shallow linguistic analysis of texts [42] with the exploitation of tags and some knowledge about the domain at hand. Such approaches avoid the efficiency problem involved in deeper text understanding, but nevertheless capture more of the semantics of the available sources than pure tagging-based approaches.

In the future, more and more sources will be multimedia sources that bring up new problems with respect to information extraction. Methods for extracting information from such multimedia information sources are still an open and interesting research problem. Information brokering [49] presents a first approach to these problems.

REFERENCES

[1] Serge Abiteboul. Querying semistructured data. In *Proceedings of the ICDT*, pp. 1–18 (1997).

[2] Serge Abiteboul, Dallan Quass, Jason McHugh, Jennifer Widom, and Janet Wiener. The Lorel Query Language for Semistructured Data. *Journal of Digital Libraries*, **1**(1) (1996).

[3] B. Adelberg. NoDoSE – a tool for semi-automatically extracting structured and semistructured data from text documents. Technical report, Computer Science Department, Northwestern University, Evanston, IL, `http://www.cs.nwu.edu/~adelberg/nodose/nodose.html` (1998).

[4] Paolo Atzeni and Giansalvatore Mecca. Cut and paste. In *Proceedings of the 17th ACM SIGACT/SIGMOD/SIGART Symposium on Principles of Database Systems*, pp. 144–153 (1997).

[5] Carlo Batini, Maurizio Lenzerini, and Shamkant B. Navathe. A comparative analysis of methodologies for database schema integration. *Computing Surveys*, **18**(4):323–364 (1986).

[6] Joachim Biskup and Bernhard Convent. A formal view integration method. In *ACM SIGMOD Proceedings*, pp. 398–407 (1986).

[7] C. Cardie. Empirical methods in information extraction. *AI Journal*, **18**(4):65–79 (1997).

[8] Mal Castellanos, Felix Saltor, and Manuel Garcia-Solaco. Semantically enriching relational databases into an object oriented semantic model. In *Proceedings of the International Conference on Database and Expert Systems Applications (DEXA)*, pp. 125–134 (1994).

[9] Christine Collet, Michael N. Huhns, and Wei-Min Shen. Resource integration using a large knowledge base in Carnot. *IEEE Computer*, **24**(12):55–62 (1991).

[10] I. Dagan and S. Engelson. Sample selection in natural language understanding. In *Connectionist, Statistical and Symbolic Approaches to Learning for Natural Language Processing*. Springer (1996).

[11] Dublin Core Metadata. `http://purl.oclc.org/metadata/dublin_core/`.

[12] S. Decker, M. Erdmann, D. Fensel, and R. Studer. Ontobroker: Ontology based access to distributed and semi-structured information. In *Semantic Issues in Multimedia Systems*. Kluwer Academic, Boston, MA (1999).

[13] Bertin Klein Dream and Peter Fankhauser. Error tolerant document structure analysis. *International Journal on Digital Libraries*, **1**(4):344–357 (1997).

[14] Daniel Dreilinger and Adele E. Howe. Experiences with selecting search engines using metasearch. *ACM Transactions on Information Systems*, **15**(3):195–222 (1997).

[15] M. Erdmann and R. Studer. Ontologies as conceptual models for XML documents. In *12th Workshop on Knowledge Acquisition, Modeling and Management (KAW'99)*, Banff, Canada (1999).

[16] E. A. Fox. Development of the coder system: A testbed for artificial intelligence methods in information retrieval. *Information Processing & Management*, **23**(4):341–366 (1987).

[17] D. Freitag. Information extraction from html: Application of a general machine learning approach. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI)*, Madison, WI (1998).

[18] N. Fuhr. Probabilistic datalog - a logic for powerful retrieval methods. In *Research and Development in Information Retrieval*, pp. 282–290 (1995).

[19] J. Geller, Y. Perl, and E. J. Neuhold. Structural schema integration with full and partial correspondence using the dual model. *Information Systems*, **17**(6):443–464 (1992).

[20] Janusz R. Getta. Translation of extended entity-relationship database model into object-oriented database model. In *DS-5*, pp. 87–100 (1992).

[21] Charles F. Goldfarb. *The SGML Handbook*. Oxford University Press (1990).

[22] Charles F. Goldfarb and Paul Prescod. *The XML Handbook*. Prentice Hall Computer Books (1998).

[23] The governments of the world. At
`http://www.louisville.edu/library/ekstrom/govpubs/international/intgov.html`.

[24] U. Hahn and K. Schnattinger. Towards text knowledge engineering. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI)*, Madison, WI (1998).

[25] Joachim Hammer, Hector Garcia-Molina, Junghoo Cho, Arturo Crespo, and Rohan Aranha. Extracting semistructured information from the web. In *Workshop on Management of Semistructured Data*, pp. 18–25 (1997).

[26] J. Heflin, J. Hendler, and S. Luke. Reading between the lines: Using SHOE to discover implicit knowledge from the web. In *AAAI'98 Workshop: AI and Information Integration*, Madison, WI (1998).

[27] C.-N. Hsu. Initial results on wrapping semistructured web pages with finite-state transducers and contextual rules. In *AAAI'98 Workshop: AI and Information Integration*, Madison, WI (1998).

[28] Gerald Huck, Peter Fankhauser, Karl Aberer, and Erich J. Neuhold. JEDI: Extracting and Synthesizing Information from the Web. In *COOPIS 98* (1998).

[29] J. Jannink, V. Pichai, D. Verheijen, and G. Wiederhold. Encapsulation and composition of ontologies. In *AAAI'98 Workshop: AI and Information Integration*, Madison, WI (1998).

[30] J.-T. Kim and D.I. Moldavan. Acquisition of linguistic patterns for knowledge-based information extraction. *IEEE Transactions on Knowledge and Data Engineering*, **7**(5):713–724 (1995).

[31] Wolfgang Klas, Peter Fankhauser, Peter Muth, Thomas Rakow, and Erich J. Neuhold. *Database Integration using the Open Object-Oriented Database System VODAK*, chapter 14: Object Oriented Multidatabase Systems: A Solution for Advanced Applications. Prentice Hall, Englewood Cliffs, N.J. (1996).

[32] Craig A. Knoblock, Steven Minton, Jose Luis Ambite, Naveen Ashish, Pragnesh Jay Modi, Ion Muslea, Andrew G. Philpot, and Sheila Tejada. Modeling web sources for information integration. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI)*, Madison, WI (1998).

[33] John Larmouth. *ASN.1 Complete*. Open System Solutions (1999).

[34] W. Litwin, L. Mark, and N. Roussopoulos. Interoperability of multiple autonomous databases. *Computing Surveys*, **22**(3):267–293 (1990).

[35] Victor M. Markowitz and Arie Shoshani. On the correctness of representing extended entity-relationship structures in the relational model. In *ACM SIGMOD Proceedings*, pp. 430–439 (1989).

[36] Ka Fai Yau Michael Rys. Data extraction from dynamic web sites: Combining crawling and extraction. In *8th International World Wide Web Conference* (1999).

[37] Microsoft, University of Edinburgh. *XML-Data reduced* (1998).

[38] G. Miller. Wordnet: A lexical database for english. *Communications of the ACM*, **38**(11):39–41 (1995).

[39] R. J. Miller, Yannis E. Ioannidis, and Raghu Ramakrishnan. Schema equivalence in heterogeneous systems: bridging theory and practice. *Information Systems*, **19**(1):3–31 (1994).

[40] I. Muslea. Extraction patterns for information extraction tasks: a survey. In *AAAI'99 Workshop Machine Learning for Information Extraction*. AAAI Press (1999).

[41] I. Muslea, S. Minton, and C. Knoblock. Learning wrappers for semi-structured, web-based information. In *AAAI'98 Workshop: AI and Information Integration*, Madison, WI (1998).

[42] G. Neumann and S. Schmeier. Combining shallow text processing and machine learning in real world applications. In *IJCAI-99 Workshop on Machine Learning for Information Filtering*, Stockholm, Sweden (1999).

[43] Christophe Nicolle, Djamal Benslimane, and Kokou Ytongnon. Multi-data models translations in interoperable information systems. In *CAiSE*, pp. 176–192 (1996).

[44] The Oxford English Dictionary. `http://www.oed.com/`.

[45] L. Page and S. Brin. The anatomy of a large-scale hypertextual web search engine. `http://google.stanford.edu/~backrub/google.html` (1998).

[46] Yannis Papakonstantinou, Hector Garcia-Molina, and Jennifer Widom. Object exchange across heterogeneous information sources. In Philip S. Yu and Arbee L. P. Chen, editors, *Proceedings of the Eleventh International Conference on Data Engineering, March 6-10, 1995, Taipei, Taiwan*, pp. 251–260. IEEE Computer Society (1995).

[47] E. Riloff. Automatically constructing a dictionary for information extraction tasks. In *11th Annual Conf. on Artificial Intelligence (AAAI'93)* (1993).

[48] A. Sahuguet and F. Azavant. Building light-weight wrappers for legacy web data sources using W4F. In *Proceedings of the 25th International Conference on Very Large Databases (VLDB)* (1999).

[49] A. Sheth, V. Kashyap, and T. Lima. Semantic information brokering: How can an multi-agent approach help? In *Third International Workshop CIA-99 on Cooperative Information Agents*, Uppsala, Sweden (1999).

[50] S. Soderland. Learning information extraction rules for semi-structured and free text. At `http://www.cs.washington.edu/homes/soderlan` (1998).

[51] S. Soderland, D. Fisher, J. Aseltine, and W. Lehnert. Inducing a conceptual dictionary. In *14th Int. Joint Conf. on Artificial Intelligence (IJCAI'95)* (1995).

[52] Stefano Spaccapietra and Christine Parent. View integration: A step forward in solving structural conflicts. *TKDE*, **6**(2):258–274 (1994).

[53] Stefano Spaccapietra, Christine Parent, and Yann Dupont. Model independent assertions for integration of heterogeneous schemas. *VLDB Journal*, **1**(1):81–126 (1992).

[54] Beth Sundheim, editor. *Proc. of the 6th Message Understanding Conference*. Morgan Kaufmann Publishers (1995).

[55] U.S. National Library of Medicine (NLM). Search MEDLINE: PubMed and Internet Grateful Med. `http://www.nlm.nih.gov/databases/freemedl.html`.

[56] W3C Candidate Recommendation. *Resource Description Framework (RDF) Schema Specification 1.0* (2000).

[57] W3C Proposed Recommendation. *Resource Description Framework (RDF) Model and Syntax Specification* (1999).

[58] W3C Recommendation. *Extensible Markup Language (XML) 1.0* (1998).

[59] W3C Working Draft. *Extensible Stylesheet Language (XSL) Specification* (1999).

[60] Ling-Ling Yan and Tok Wang Ling. Translating relational schema with constraints into oodb schema. In *Interoperable Database Systems (DS-5) IFIP WG2.6 Database Semantics Conference*, pp. 69–85 (1992).