Information Extraction with AI Planning

Csaba Dezsényi, Tadeusz Dobrowiecki, Tamás Mészáros

Budapest University of Technology and Economics, Department of Measurement and Information Systems, Budapest, Hungary
{dezsenyi, dobrowiecki, meszaros}@mit.bme.hu

Abstract: Autonomous information systems, which answer complex queries by extracting information from electronic sources written in natural language, must be designed for maximum flexibility and adaptivity. This paper proposes an approach that is based upon the library of elementary document processing modules organized adaptively from query to query into an information-processing network. This network is both a tool of the scheduling and controlling the execution of the modules and a framework for the semantic fusion of heterogeneous information chunks. In the present paper the algorithmic background of the network design is presented in detail.

Keywords: document processing, planning, information extraction, adaptivity

1 Introduction

The rapid and wide spread of the Internet resulted in a huge amount of online information. However, the exploitation of the available information requires endless human effort. The investigation of the automated information processing acquired thus a significant role in the past decade, both in the business and in the sciences. Related software systems and applications contain more and more of the built-in intelligence, on the other hand the traditional solutions slowly become ineffective and labor-intensive.

One of the most important research topics is the Information Extraction (IE). Its purpose is to automatically extract relevant pieces of information from documents containing human-written texts [1]. Various document analysis and processing techniques related to the IE are intensively used in an ever growing spectrum of applications. We can find them in simple information processing tools (e.g. spam filters, personal web assistants), but also in large, corporation-wide solutions (e.g. knowledge management and decision support systems, customer relation management solutions, knowledge intensive search engines). IE methods and algorithms are based on several distinct approaches, like statistics [2], pattern fitting [3], machine learning [4], natural language processing [5], and many others.

To extract relevant information from a document and achieve significant performance it is not sufficient to use a single algorithm, multiple processing steps should be applied. To see the reasons consider the aim of developing an application that retrieves economic articles from news portals, extracts simple facts about companies and persons, and puts them into the application's knowledge base. Several document processing tasks are required to extract such information. The article should be extracted from the HTML page, the word and sentence boundaries marked, the company and person names identified in the text, morphological analysis and a sentence parsing to extract simple subject-predicate-object triplets performed, finally e.g. the statistical analysis should be used to calculate relevancies for controlling local search. It is relatively simple to implement and execute such steps separately. These modules, however, should work in concert to produce adequate results for the complex extraction tasks.

The R&D of such complex IE systems was done in isolation so far. The document processing subsystems were mostly based on individual applications, with different architectures, datamodels and working mechanisms. No guidelines, de facto rules existed that could assist the fusion of document processing modules into a whole integrated system. The purpose of the present research is to design a general document analysis and processing framework that can facilitate the creation of arbitrary complex IE applications.

The general principle is to analyze a complex query, to select special purpose document processing modules and to organize them into a network accordingly to the structure of the query, then to execute this network adaptively fusing on the way the particular information chunks into a complex answer. To this aim a suitable formalism and context independent framework has been designed, which involves general principles, architectural considerations, and methodological guidelines [7]. Suitable data models and interfaces have been created that enable the usage of the framework in a wide range of applications.

There are two key challenges in the research: one is the design of proper datamodels that are suitable for fusing together the isolated results of different processing modules, producing thus semantically linked and coherent result of the whole processing operation [7]. The second is the automated planning of the processing schema that determines the running sequence and collaboration of the processing modules. In this paper the algorithm for the design of the automated document processing schema, based on AI planning techniques, is presented.

2 The Information Extraction Flow

Generally information extraction can be divided into the retrieval, the document processing and the extraction steps (Fig. 1). First, a document has to be retrieved from the source environment of an application. The result is an initially structured document. In the simplest case, an application may require only text files, iteratively read from a local directory. A more autonomous solution would apply an intelligent retrieval agent that can search the Internet for relevant documents [6]. The retrieval phase can also include format conversion, document merging or splitting, and other required preprocessing.

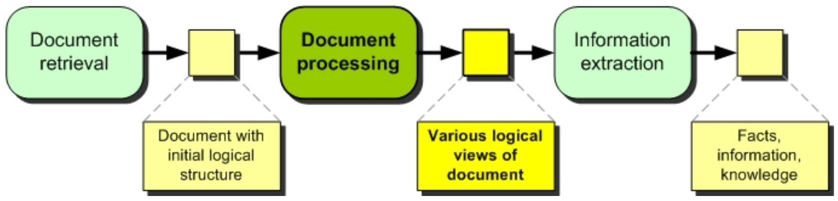

Document retrieval
Document processing
Information extraction
Document with initial logical structure
Various logical views of document
Facts, information, knowledge

Figure 1
The main phases of the information extraction

The second step is to process the document in several ways to produce various semantically structured representations of the original source, which we call views. These views contain the demanded chunks of information and yield the input to the third step, where the information is extracted and stored in a local knowledge repository of the application.

In traditional solutions, the outlined subsystems are hard coded and specific to the application. Our aim is to develop a general framework that is able to receive arbitrary document analyzer and processing modules (DA - document analyzer) and datamodel configurations (views) and to adaptively solve various IE task for an application. The model of the whole information extraction process, initiated and concluded in the application is presented in Fig. 2.

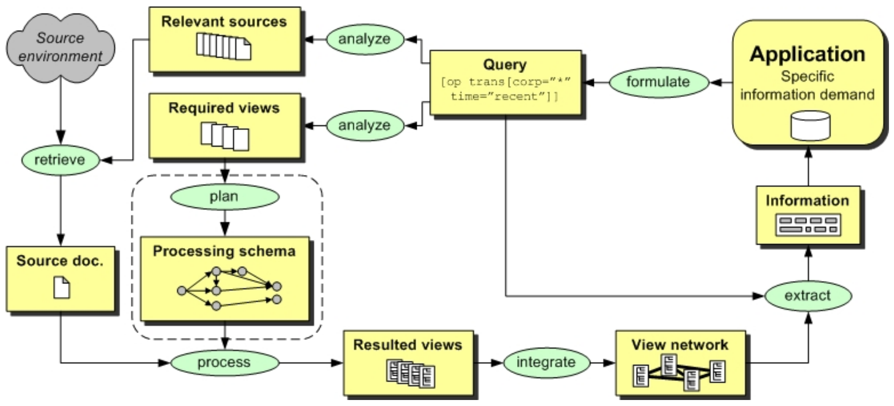


Figure 2
Query based adaptive information extraction

A particular application represents well-defined information demands that can be formulated as an abstract (explicit or implicit) query. By the analysis of this query we can determine the required logical representations (views) of the source documents that are necessary for the extraction. The query may also determine the possibly relevant set of information sources for the retrieval process.

The framework contains several DAs that can be applied to produce views. DAs can also depend on each other, thus the flow of the processing is not trivial. With the list of required views, however, the appropriate processing schema can be planned automatically. After retrieving a new source document, the framework will apply the planned processing operations to it. The results are the required logical views. It is important that the results of the independent DAs should be semantically linked and coherent, thus the framework must ensure the proper integration of the views. The semantically integrated set of resulting views is called the view network (see Fig. 2) [7].

3 Views and Document Analyzer Modules

Traditional document processing modules are usually complete processing units. In our approach they are basic building blocks that can be applied in various ways to construct complex and adaptive processing schema. Therefore, the modules should be designed for reusability. They also should be able to reuse the partial results of each other, if necessary or practical. They should provide proper methods for fine tuning and should be sufficiently scalable. Based on these objectives we have proposed an abstract document analyzer model. This model ensures that the framework can handle the processing problems uniformly, without referring to concrete module implementations. It is also responsible for producing semantically linked and coherent results. This model is presented in detail in [7], here we describe only that part which is essential in the presentation of the planning algorithm.

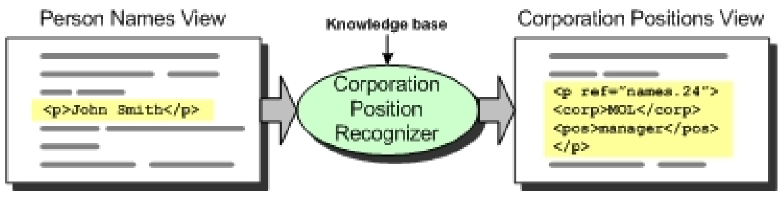


Figure 3
Example document processing module with its input and output views

The task of DA is to recognize certain elements in its input document view and transform them to its output view (Fig. 3). The resulting view of a DA of a particular type is a kind of information projection of the source that contains the transformed information in a new, semantically structured form. The views are typed, where the type defines what kind of information is marked in the view and yields its structure by suitable definitions. DAs can operate on one or more views. The input to the framework is an initial view that contains the original source document after the retrieval, with its initial structure, while the set of all the created views in the framework is the complete result of a document processing. The datamodel that contains all the semantically linked views generated from one source document is the view network.

The views are implemented as special XML documents, because it perfectly fits the demands of carrying semantically marked information. XML is also a widespread standard, thus we could benefit from the application of various standard freely available tools (e.g. DOM parser, XSchema validator, XPath and XSLT processor, etc. [10]).

4 Planning the Document Processing Schema

The task of the framework is to produce the required views (defined by the application) by applying proper DAs to a source document. The challenge is to design an effective planning algorithm that can produce a processing schema able to solve the given task. The algorithm should draw on the available DAs and the incoming document features. The main objective is to make it as autonomous as possible, to deploy the framework to different application environments with minimum effort on manual configurations (the configuration of the framework should be data-driven, not procedural, which facilitates the application independence). Moreover, the set of the available DAs can also change during the lifetime of an application, e.g. by introducing new modules for increased performance or by implementing new features.

Selecting the required DAs for execution and planning of the right running sequence is not trivial, because some components may depend on the outputs of the others, some can work on several views, or the same result could be produced by applying different processing components. The basis of the approach is the adaptation of the philosophy of the standard AI planning techniques (i.e. STRIPS based partially ordered planning [8]). The application of such technique is plausible due to the analogy of the problem. However, the adaptation is far from trivial, because there are essential differences also. In the following we introduce first the application of the AI planning to the IE, then we discuss the special issues in detail.

4.1 Adopting Standard AI Planning

Concepts central to the standard STRIPS-based partially ordered AI planning can be matched easily with those of the framework (see Table 1).

Table 1
Standard AI planning concepts vs. the concepts of the framework

Standard AI Planning	Document Processing Framework
State of the environment	State of the view network – availability of views with specific type and content
Operator	DA that changes the state of the view network by producing new views
Initial state	Initial view - the source document in an initial structure
Goal state	Existence of specific views required by the application
Precondition	Type definitions of input views of a module required for its execution
Effect	Type definitions of output views of a module produced after its execution
Action description	Implementation of the module, which can be executed by the framework
Partially ordered plan	Final document processing schema

The aim of the processing is posed as a number of views containing the demanded information. Then the graph of the plan is constructed regressively by matching still unfulfilled preconditions to the effects of the newly introduced operators until there are no unfulfilled preconditions in the graph [8]. However, AI planning cannot be adopted blindly. Switching over from the general planning domain to the information extraction introduces two essential simplifications and one essential difficulty. The former is discussed shortly below, whilst the latter is detailed in the next chapter.

A typical difficulty in traditional planning is when one operator deletes the already satisfied preconditions required by the other. This problem can be handled by so-called protected links between the effects and the satisfied preconditions (see more details in [8]). An advantageous feature of the framework is that the effects of the operators can only contain positive literals. DAs can incrementally produce new views, but deleting them is against the philosophy of information extraction. Another simplification is that there is no need for the linearization of the partially ordered plan, because the modules can be run in parallel.

4.2 Handling Conflict Situations

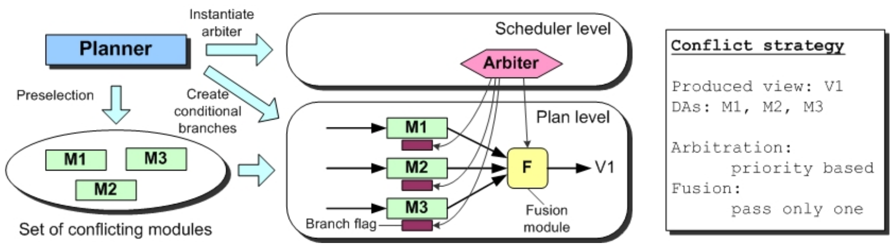
Significant problem is handling conflicting situations. A common demand in the state-of-the-art IE applications is to use hybrid solutions in document processing phases. Typical example is the combination of statistical and symbolical methods, aggregating the advantages of the both approaches. We can also have a reliable tool for previously known cases in one processing task, but we also want to handle unknown cases with a heuristic tool. Still other examples can be the language, and domain dependent processing modules. There are cases when more DAs may be used for a processing task, and the optimal selection mechanism would also depend on the kind of the conflict. In such case the problem should be analyzed in detail, considering the possibly occurring cases, and the planning algorithm should be extended with the capability of handling such situations. After introducing the main principle, we present the algorithm through a simple example scenario.

Conflict situation occurs, when there is more than one DA for producing the required view. A standard AI planner would handle such situations with a simple non-deterministic selection, because the logical descriptions of the effects are the same. However, DAs with the same effect description can produce the same type of views with significantly different quality of the content. It is impossible or impractical to formally describe such complex difference in the produced content, thus we cannot handle it solely in the planning time. In addition, some cases would require the selection of more than one conflicting DA and to integrate the contents of the results into a single view. Therefore, in our approach new elements are introduced in the planning algorithm, in the execution scheduler, and in the plan as special additional operators (Fig. 4).

If the planner takes notice that more than one module can be found for producing the required type of view, it can make a preselection by evaluating the available metadata definitions of the modules. E.g. it can recognize that certain modules are offline, or outdated. This first rough selection is used to reduce the number of conflicting modules at the planning time, if possible, reducing this way the branching factor, which can pop up in the execution time.

The decision about the remaining set of modules can only be made during the execution. Consequently we have to introduce special operators into the plan that enable the evaluation of the results after running the modules. The planner inserts all of the mentioned modules into the plan, but marks them with conditional branch flags, and links their outputs to a ***fusion module***. Each conflicting module forms one branch marked by a binary flag. If the corresponding flag is true, the branch will be executed in the execution phase, when its preconditions are fulfilled. Flags are controlled by ***arbiters***, instantiated by the planner for each conflicting set of modules. An arbiter controls the running of the conflicting modules at the execution time by re-setting the corresponding branch flags. After one or more branches are executed and the results evaluated, arbiter can modify the flags to rerun the modules in different configuration if necessary.

The task of fusion module is to merge the incoming views (all of the same type) into a single view. The algorithm is based on the comparison of the information elements in the views. If the arbiter strategy enables only one valid view, the fusion module simple passes it through. However, if the strategy is to integrate several views, the merging is necessary and not trivial.



```
Conflict strategy

Produced view: V1
DAs: M1, M2, M3

Arbitration:
      priority based
Fusion:
      pass only one
```

Figure 4

(a) Main elements of the algorithm, (b) Strategy definition for the conflicting modules

Particular control strategy built-in into an arbiter and fusion module is associated with the particular set of the DAs (defined as a kind of macro operator, Fig 4b). Typical choice can be the priority-based selection of the branches, or executing all branches together, merging the results (see later).

After the conflicting modules have been put into the plan, the arbiter and the fusion modules initialized, and the modules are marked with the corresponding branch flags, the planner takes the next unsatisfied precondition and attempts to find a module that can produce the required view. The only additional feature is that the planner has to spread backward the conditional flags properly to the preceding modules in the branches (see details in the example later).

4.3 Example Scenario

As example application, consider a system that extracts economical facts about relevant persons and companies, e.g. a firm is liquidated, or a person is appointed to a new job, etc. The underlying framework's configuration contains the following required views and processing modules (Table 2 and 3).

Table 2

View types in the example

View	Description
H	Original HTML source page with metadata
A	Extracted textual article, segmented e.g. as title, authors, date, and content
T	Tokenized text, textual parts segmented to sentences, words, and other tokens
P	Part of speech of words
S	Word stems
N	Recognized names (e.g. company and person names)
C	Recognized concepts (e.g. defined in the application's ontology)
E	Parsed sentence trees
F	Extracted fact candidates

Table 3

Processing modules in the example

Module	**Precondition** (view)	**Effect** (view)	**Description**
W	H	A	**Web-wrapper**: has predefined rules for known Internet sources, e.g. we can extract the articles from a known portal with semantic structure [9]
CE	H	A	**Content extractor**: tries to extract the content from a webpage, cutting off menus, advertisements, etc.
TO	A	T	**Tokenizer**: marks word and sentence boundaries
ST	T	S	**Stemmer**: finds lemmatized word forms
PO	T	P	**Part of speech tagger**: finds part of speech of words
NL	S	N	**Lexicon-based named entity recognizer**: extracts relevant names using predefined lexicons
NH	S	N	**Heuristic named entity recognizer**: extracts relevant names using patterns and algorithms
CO	S	C	**Concept finder**: marks concepts found in the application's knowledgebase
SP	P	E	**Sentence parser**: analyzes structure of sentences
FP	E N	F	**Fact extractor**: based on fitting simple patterns on analyzed tokens
FS	P N C	F	**Fact extractor**: collects subject-predicate-object triplets based on syntactic structure
S	-	H	**Start**: creates initial state (initial view)
G	F	-	**Goal**: defines the required views

The final plan can be seen in the Fig. 5. Fusion modules are rounded squares, and branch conditions are indicated be-low the DAs There are three conflicting DA sets, each one connected to its fusion module (**F_A**, **F_N**, **F_F**). Each conditional branch is marked with proper flags (e.g. **A(w)** and **A(ce)** at the conflict of **W** and **CE** due to the same effect **A**). If a conditional branch has a preceding DA, the flag has to be propagated backward. See e.g. the **CO** module, which receives **F(fp)** flag from **FP** and will be executed if the **A_F** arbiter decides to set the **F(fp)** flag on. If a module forks out into more branches (see e.g. **ST**, or **PO**), it receives the disjunction of the corresponding branch flags. A module should run if at least one of the branches is selected. However, if it receives flags of all of the following branches, the flags are redundant and can be deleted (indicated with crosses in the Fig. 5).

Arbiter mechanisms are different in all three conflict situations. The **A** branches typically should run with a priority mechanism: if an HTML page has an associated wrapper definition, then **W** should run, otherwise **CE** tries to extract the textual content with a heuristic algorithm. Thus, arbiter $\mathbf{A_A}$ first tries to execute **W** by setting **A(w)** true and **A(ce)** false. If **W** doesn't yield result, **A(ce)** will be set true, which enables the execution of **CE**. Arbiter $\mathbf{A_N}$ will execute each branch together, and $\mathbf{F_N}$ will merge the results, because that way the final document view will contain the most of the recognized named-entities. Arbiter $\mathbf{A_F}$ can also execute each branch in parallel.

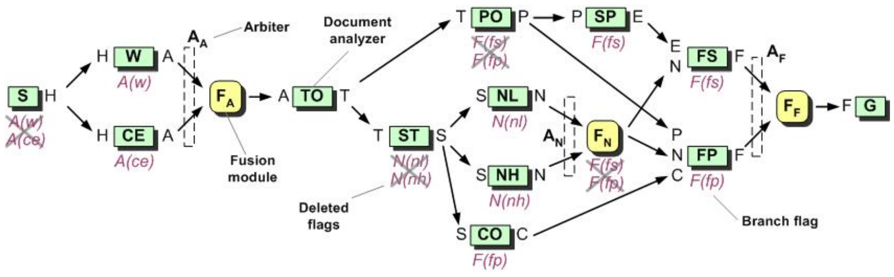


Figure 5

The planned document processing schema for a particular task

When the planner stops with a complete partially ordered plan, we obtain the final schema presented in Fig. 5. If successive branches would be required in the plan, the spreading of the flags can also be done simply, leading however to conjunctions. It means that a module before two successive branches will run only if each branch is selected for the execution.

Conclusions and Further Work

We presented a novel solution for information extraction, where we have developed an adaptive document analysis and processing framework. The proposed approach and algorithm show two principal advantages. Firstly, it fits the standard AI planning algorithms, where arbitrarily complex and numerous conditional branches can be handled effectively. Secondly, the method precisely separates the algorithmic steps of the decision making and the result fusion. Furthermore, it enables the usage of practical strategies for the real cases.

An additional demand in the framework is the disjunctive preconditions, which would model the situation, when a DA is able to process different type of input views. A sentence and word tokenizer e.g. can run on an *article* and also on a *scientific-paper* view. These kinds of conflict situations also can be handled with the outlined mechanism.

Currently the prototype version of the framework is being implemented, with a number of document processing modules. This prototype will serve to tune further the basic algorithm, and also to experiment with the suitable heuristic arbitration strategies. Current document processing modules are oriented toward short electronic news in Hungarian, however the architecture and the functioning of the framework are language and context independent.

References

[1] Cunningham, H.: Information Extraction – a User Guide. Research Memorandum CS–97–02, Department of Computer Science, University of Sheffield, January (1997)

[2] Neto, J. L., Santos, A. D., Kaestner, C. A., Freitas, A. A.: Document Clustering and Text Summarization. In Proc. of the 4th Int. Conf. on Practical App. of Knowledge Discovery and Data Mining, pp. 41-55, London, UK (2000)

[3] Hammer, J., Breunig, M., Garcia-Molina, H., Nestorov, S., Vassalos, V., Yerneni, R.: Template Based Wrappers in the TSIMMIS System. In Proc. of the 23rd ACM SIGMOD Int'l Conf. on Management of Data, Tucson, Arizona (1997)

[4] Kushmerick, N., Thomas, B.: Adaptive Information Extraction: Core Technologies for Information Agents. In Intelligent Information Agents R&D in Europe: An AgentLink perspective, Lecture Notes in Computer Science 2586, Springer, (2003)

[5] Mitkov, R. (ed.): The Oxford Handbook of Computational Linguistics. Oxford University Press, Oxford (2003)

[6] Varga, P., Mészáros, T., Dezsényi, Cs., Dobrowiecki, T.: An Ontology-based Information Retrieval System. In Proc. of the IEA/AIE-2003, Laughborough, UK, Lecture Notes in Computer Science vol. 2718 Springer-Verlag (2003)

[7] Dezsényi, Cs., Mészáros, T. Dobrowiecki, T.: Parser Framework for Information Extraction. Proc. of the EUROFUSE Workshop on Data and Knowledge Engineering, Sept 22-25, Warsaw, Poland (2004)

[8] Russell, S., Norvig, P.: Artificial Intelligence. A Modern Approach. Prentice Hall Inc. (1997)

[9] Kuhlins, S., Tredwell, R.: Toolkits for Generating Wrappers. Net.ObjectDays-2002, Erfurt, Germany, October (2002)

[10] Information about several XML standards can be found on http://www.w3c.org