Aggregator: A Knowledge based Comparison Chart Builder for e-Shopping

F. Kokkoras, N. Bassiliades and I. Vlahavas
Department of Informatics
Aristotle University of Thessaloniki, Greece
{kokkoras, nbassili, vlahavas}@csd.auth.gr

Abstract

Although there are many on-line stores, where potential e-shoppers can purchase desired products by visiting a few web pages, finding the right product to purchase is quite a tedious task. On-line stores usually offer a limited set of ill-described products, presented in a way that prevents side-by-side comparison shopping. In this chapter, we present a comparison chart building approach that is based on information extraction wrappers. The novelty of our approach consists of the usage of the Conceptual Graphs knowledge representation and reasoning formalism, which naturally supports both the wrapper induction and the wrapper evaluation tasks through the generalization, specialization and projection operations. In addition, the graphical representation of Conceptual Graphs makes them a proper technology for creating visual, wrapper management programming environments. Finally, we present the Aggregator, a prototype system based on our approach, that allows the user to quickly and visually train information extraction wrappers and use them to built comparison charts of arbitrary detail, gathering information about similar products from multiple known sites.

1 Introduction

Most internet stores selling certain types of products, usually offer a limited set of brand names and for each brand name, a limited set of products. In addition, the design of such e-commerce sites is strongly influenced by retailers whose only goal is to sell as many products as possible to the users that visit their site. As a result, such sites follow a fixed representation for the products offered and put more emphasis on the price, less emphasis on the complete presentation of the features of the product, and unfortunately, they discourage side-by-side comparison shopping. Moreover, presenting various products, they put emphasis on just a few strong features and they don't mention the weak ones. Although such e-shops are valuable for the final purchase transaction, they fail to service the non-informed customer, that is, the potential buyer that has no clear picture of what exactly to buy from the available alternatives.

Such an information need from the customer side, can be usually covered by browsing to the product's brand site where detailed specification pages about their products can be found. The negative aspect of this approach is the huge amount of time that is required by the buyer to create a clear picture of what are the advantages and disadvantages of the available products. Considering that there are many brands making the desired product and that each of them offers many models, browsing at so many specification pages is a time consuming task. To make things worst, comparing the different models can be done, manually only, on paper or by copying and pasting information to another application, such as a spreadsheet. Even for the experienced web user this work-load discourages such a task.

The discussion above makes clear that there is a need for software tools that allow the as effortless as possible creation of comparison shopping charts by gathering product specification information from various known sites. This is not an information retrieval task but rather an information extraction one. A web search engine can probably help to locate an information resource but is unable to process that resource, extract feature-value pairs and integrate that information into a singe comparison table.

In the recent years, various researchers have proposed methods and developed tools towards the web information extraction task, with the buzzword of the field being the term wrapper. A *wrapper* (or extraction rule) is a mapping that populates a data repository with implicit objects that exist inside a given web page. Creating a wrapper, usually involves some training (wrapper induction - [31]) by which the wrapper learns to identify the desired information. Unlike Natural Language Processing (NLP) techniques that rely on specific domain knowledge and make use of semantic and syntactic constraints, wrapper induction mainly focuses on the features that surround the desired information (delimiters). These features are usually the HTML tags that tell a web browser how to render the page. In addition, the extraction of typed information like addresses, telephone numbers, prices, etc., is usually performed through extensive usage of regular expressions (Figure 1). Regular expressions are textual patterns that abstractly, but precisely, describe some content. For example, a regular expression describing a price in euros could be something like "€ \d".

Besides regular expressions, there are two major research directions in wrapper induction. The first and older one, treats the HTML page as a linear sequence of HTML tags and textual content ([2], [26], [35], [37]). Under this perspective, a wrapper generation is a kind of substring detection problem. Such a wrapper, usually includes delimiters in the form of substrings that prefix and suffix the desired information. These delimiters can be either spotted to the wrapper generation program by the user (supervised learning) or located automatically (unsupervised learning). The former method usually requires less training examples but should be guided by a user with a good understanding of HTML. The latter approach usually requires more training examples but can be fully automated.

A regular expression, identifying *font* HTML tags.

Extraction Rule: (?i)<FONT size=(["']?)([+-]?\d+)\1>

Source: ...$\boxed{\texttt{<FONT size="+2">}}$ hello </FONT> $\boxed{\texttt{<FONT size=1>}}$ world </FONT>...

A linear wrapper extracting a digital camera model name from an HTML snippet.

Extraction Rule: skipto(<B>), extractUntil(X, </B>)

Source: ...<P>New model: <B>$\boxed{\text{Nikon Coolpix 3200}}$</B></P>...

A hybrid wrapper as a path expression (tree wrapper) combined with a regular expression "€\d", that extracts prices in euros from HTML table cell tags.

Extraction Rule: *.table.*.td(X, "€\d")

Source: ....<TABLE><TR><TD>Canon S300</TD><TD>$\boxed{\text{€ 450.00}}$></TD>.....

Figure 1: Typical expressions of wrappers of various technologies and their extracted result (framed text)

As the Internet technologies emerge, a new breed of wrapper induction techniques appeared ([8], [12], [30]), that treat the HTML document as a tree structure, according to the Document Object Model (DOM) [18]. Basically, such a tree wrapper uses path expressions to refer to page elements that contain the desired information (Figure 1). Tree wrappers seem to be more powerful that string wrappers. Actually, if input documents are well structured and tags at the lowest level does not contain several types of data, then a string wrapper can always be expressed as a tree wrapper [36].

Thanks to the advanced tools that are available for web page design, HTML pages are nowadays highly well-formed, but at the same time the content is more decorated by using more HTML tags. As a result, although approximate location of desired information is relatively easy thanks to tree wrappers, extraction of the exact piece of information requires regular expressions or even NLP (Figure 1). As a result, such hybrid approaches are quite popular.

In general, wrapper induction technology demonstrates that shallow pattern matching techniques, which are based on document structural information rather that linguistic knowledge, can be very effective. Until the semantic web [7] becomes a common place, information extraction techniques will continue to play an important role towards the informed customer concept.

In the comparison chart building problem, extracting and integrating information from heterogeneous web sources requires more than one wrappers. Variety in the way information is encoded and presented requires the cooperation of individual information extraction agents that are specialized for certain pieces of information and web sources. Creating, coordinating and maintaining a large number wrappers is not a simple task though. A crucial factor that can alleviate this burden is the way wrappers are encoded and trained. Having to modify an ill-described wrapped that ceased to work efficiently due to certain reasons, is much more difficult than modifying a wrapper described in a human friendly way. This need is becoming critical as more non-expert users are adapting information extraction technologies for personalization and information filtering. Visual tools that allow the easy creation of wrappers ([1], [4], [20], [27], [32]) and declarative languages ([4], [29], [32]) for wrapper encoding is the current established trend.

In this chapter, we present a knowledge based approach on comparison chart building from heterogeneous, semi-structured sources (product specification web pages). We propose the usage of the Conceptual Graphs (CGs) knowledge representation and reasoning formalism to train and describe information extraction wrappers. CGs naturally supports the wrapper induction problem as a series of conceptual graph (CG) generalization and specialization operations between training examples expressed as CGs. From the other hand, wrapper evaluation corresponds to the CG projection operation. Additionally, using DOM and product related domain knowledge, as well as advanced visual tools, we turn the wrapper creation and testing problem in an effortless task. Finally, we present the Aggregator, a comparison chart builder program that is based on the proposed approach. Aggregator can be taught how to gather specification information from web pages offered by brand sites and then use this knowledge to create side-by-side feature comparison charts by mining web pages in a highly automated and accurate fashion.

The rest of the chapter is organized as following: Section 2 presents related work in the field of wrapper induction and information extraction, emphasizing in comparison shopping and visual approaches. Section 3 gives a short introduction to CGs and proposes a novel approach for wrapper training, modeling and evaluation that is based on CGs. Section 4, presents how our CG-based wrappers and domain knowledge can be used to create comparison charts from heterogeneous web sources. Section 5 outlines the Aggregator, a tool that allows to visually train and apply CG-based wrappers, and finally, Section 6 concludes the chapter and gives insight for future work.

2 Related Work

In the last few years, many approaches and related tools have been proposed to address the web information extraction problem. In the following, we give some detail about approaches that are closer to ours, in the sense that, they either exploit a tree representation of a web page ([4], [29], [32]) or use target structures that describe objects of interest and try to locate portions of web pages that implicitly conform to that structures ([1], [20], [27]). A good survey on information extraction from the web can be found in [28].

XWRAP [29] is an interactive system for semi-automatic generation of wrapper programs. Its core procedure is a three step task in which the user, first identifies interesting regions, then identifies token name and token value pairs, and finally identifies the useful hierarchical structures of the retrieved document. Each step results in a set of extraction rules specified in a declarative language. At the end, these rules are converted into a Java program which is a wrapper for a specific source. XWRAP features a component library that provides source independent, basic building blocks for wrappers and provide heuristics to locate data objects of interest.

In W4F ([32], [33]), a toolkit for building wrappers, the user first uses one or more retrieval rules to describe how a web document is accessed. Then, he/she uses a DOM representation and a web page annotated with additional information, to describe what pieces of data to extract. Finally, he/she declares what target structure to use for storing the extracted data. W4F offers a wizard to assist the user in writing extraction rules which are described in HEL (HTML Extraction Language) and denote an assignment between a variable name and a path-expression. The wizard cannot deal with collection of items, so if the user is interest in various items of the same type with the one clicked on, conditions must be attached to the path expression to write robust extraction rules.

Lixto ([3], [4]) is a system that assists the user to semi-automatically create wrapper programs by providing a visual and interactive user interface. It allows the extraction of target patterns based on surrounding landmarks, on the content itself, on HTML attributes, on the order of appearance and on semantic and syntactic concepts. In addition, it allows disjunctive wrapper definition, crawling to other pages during extraction and recursive wrapping. Wrappers created with Lixto are encoded in Elog, a declarative extraction language which uses a datalog-like logical syntax and semantics. Lixto TS [5] is an extension to the basic system aiming at web aggregation applications through visual programming.

NoDoSE [1] provides a graphical user interface in which the user hierarchically decomposes the web document, outlining its interesting regions and describing their semantics. This decomposition occurs in levels; for each one of them the user builds an object with a complex structure and then decomposes it in other objects with a more simple structure. The system uses this object hierarchy to identify other similar objects in the document. This is accomplished by a mining component that attempts to infer the grammar of the document from objects constructed by the user.

DEByE [27] is an interactive tool that allows the user to assemble nested tables (with possible variations in structure) using pieces of data taken from the sample page. The tables assembled are examples of the objects to be identified on the similar target pages. DEByE generates object extraction patterns that indicate the structure and the textual surroundings of the objects to be extracted. These patters are then fed to a bottom-up extraction algorithm that takes a target page as input, identifies on it atomic values in this page and assembles complex objects using the structure of the pattern as a guide.

In [20], an ontology based approach to information extraction is presented. The ontology (conceptual model), which is described in the Object-oriented Systems Model, is constructed prior to extraction and describe the data of interest, relationships, lexical appearance and context keywords. The extraction tool uses this ontology to determine what to extract from record-sized chunks that are derived from a web page and are cleared from HTML tags. This use of ontological knowledge enables a wrapper to "sustain" in small variations existing in similar web pages (improved resiliency) and to be able to work better with documents presenting similar information but differently organized (improved adaptivity).

Our proposed framework for wrapper creation offers very similar functionality with all of the above approaches, in the sense that it provides a visual environment for wrapper creation. There exists a major difference though in the core technology used, which, for our tool is the Conceptual Graph formalism. Our choice allow us to exploit both DOM representations of web documents (approach used in [4], [29] and [32]), as well as user defined structures that describe objects of interest (approach used in [1] and [27]). We achieve this by using CG-based generic wrapper descriptions which are detailed by the user in an interactive way, using visual tools that combine not only the DOM representation, but the browser itself. The CG formalism, naturally supports all the major steps in information extraction with wrappers, with its generalization, specialization and projection operations. In addition, CGs is a proven technology to encode ontological knowledge to provide a common schema for information integration and to improve wrapper's resiliency and adaptivity in the way [20] does. Beyond that, the representation we use provides the operations required to create a functional reasoning system. This allows the creation of dynamic ontologies, where static and axiomatic/rule knowledge co-exist [15]. For example, we can use such knowledge to create structural dependencies between two wrappers. Finally, the CG formalism has, by nature, better visualization potential. This enables our system to provide a more comprehensible wrapper representation to the end-user.

Regarding comparison shopping, one of the earliest attempts is ShopBot [19]. It focuses on vendor sites with form based search pages, returning lists of products with a tabular format. With today standards, ShopBot is quite restricted since it uses linear wrappers and focuses on highly structured pages. A commercial version of ShopBot, known as Jango, was bought by Excite.

Apart from Lixto TS [5], there are many other commercial wrapping services available on the Internet, such as Junglee (bought by Amazon), Jango, mySimon, RoboShopper and PriceGrabber. Jango and mySimon use real time information gathering from merchant sites, while Junglee pre-fetches information in a local database and updates it when necessary. All sites provide comparative shopping based on integrated information delivered from other vendor sites. Besides their unknown technology which is considered a business asset, most of these sites put emphasis on the price and provide very limited product specification information. Only PriceGrabber offers side-by-side and specification information rich, comparison charts.

3 Wrappers as Conceptual Graphs

In this section we first give a small introduction to CGs, focusing mainly on the generalization, the specialization and the projection operations which are the key ideas behind our proposed CG-Wrap model. Then we present how CGs can be used to model information extraction wrappers.

3.1 Conceptual Graphs Background

The elements of CG theory ([14], [34]) are *concept-types*, *concepts*, *relation-types* and *relations*. Concept-types represent classes of entity, attribute, state and event. Concept-types can be merged in a lattice whose partial ordering relation < can be interpreted as a categorical generalization relation. A concept is an instantiation of a concept-type and is usually denoted by a concept-type label inside a box or between "[" and "]" (Figure 2). To refer to specific individuals, a referent field is added to the concept ([table:*] - a table, [table:{*}@3] - three tables, etc). Relations are instantiations of relation-types and show the relation between concepts. They are usually denoted as a relation label inside a circle or between parenthesis (Figure 2). A relation type determines the number of arcs allowed on the relation as well as the type of the concepts (or their subtypes) linked on these arcs.

[Wrapper] ← (targetURL) ← [URL]

Figure 2: A Conceptual Graph stating that there exists a wrapper aiming at some URL

A Conceptual Graph is a finite, connected, bipartite graph consisting of concept and relation nodes (Figure 2). Each relation is linked only to its requisite number of concepts and each concept to zero or more relations. CGs represent information about typical objects or classes of objects in the world and can be used to define new concepts in terms of old ones.

The type hierarchy established for both concepts and relations is based on the intuition that some types subsume other types, for example, every instance of the concept *Table* would also have all the properties of *HTMLElement*. In addition, with a number of defined operations on CGs (canonical formation rules) one can derive allowable CGs from other CGs. These rules enforce constraints on meaningfulness; they do not allow nonsensical graphs to be created from meaningful ones. Among other operations defined over CGs, the most useful and related to the information extraction problem, are the generalization, the specialization and the projection operations.

The *generalization* is an operation that monotonically increases the set of models for which some CG is true. For example, CG$_3$ in Figure 3 is the minimum common generalization of CG$_1$ and CG$_2$. Only common parts (concepts and relations) of CG$_1$ and

CG_2 are kept in CG_3. In addition, individual concepts like [BGColor:"#FFFFFF] have become generic by removing the referent field. *Specialization* is the opposite to the generalization operation. It monotonically decreases the set of models for which some CG is true. This is achieved by either adding more parts (concepts and/or relations) to a CG, or by assigning an individual referent to some generic concept.

CG_1:	[HTMLElement: #3] ← (attribute) ← [BGColor: "#FFFFFF]
CG_2:	[HTMLElement: #9] ← (attribute) ← [BGColor: "#CCFF12] ← (parent) ← [HTMLElement: #8]
CG_3:	[HTMLElement] ← (attribute) ← [BGColor]

Figure 3: CG_3 is the minimum common generalization of CG_1 and CG_2

Projection is a complex operation that projects a CG *v* over another CG *u* which is a specialization of *v* ($u \leq v$), that is, there is a sub graph *u'* embedded in *u* that represents the original *v*. The result is one or more CGs πv which are similar to *v* but some of its concepts is possible to have been specialized by either specializing the concept type or assigning a value to some generic referent, or both.

Under the machine learning perspective, training information extraction wrappers is a combination of generalization and specialization operations that result in a model (pattern) that describes best the training instances and that can be used to detect new, unknown instances. This is similar to the generalization and specialization operations of the CG theory. A CG wrapper is the result of generalization and specialization operations over two or more training instances expressed as CGs. Moreover, applying a CG wrapper is equivalent to a projection operation of the wrapper over web page elements expressed as CGs. Based on these analogies, we present next how CGs can be used to model and train information extraction wrappers.

3.2 Modeling and Training Wrappers with CGs

The ability of CGs to represent entities of arbitrary complexity in a comprehensible way, make them a promising candidate for modeling information extraction wrappers. This perception is strengthened by the highly structured document representation which is defined by the DOM specification. This tree structure allows the easy mapping of web document elements to CG components.

```
[Wrapper]  ← (targetURL) ← [URL]
           ← (output) ← [Info]
           ← (container) ← [HTMLElement]
```

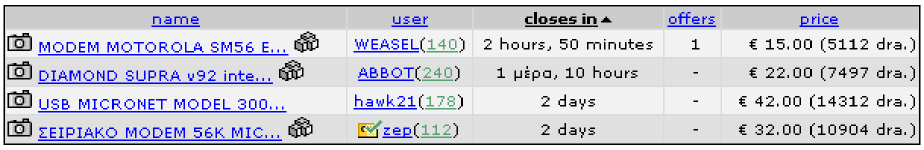
Figure 4: An abstract wrapper as a conceptual graph

In general, a wrapper accesses a page located at a specific URL, searches inside this page for some specific HTML element which is the container of the desired information and extracts that information from it. This abstract description is encoded as the CG depicted in Figure 4. In practice, such a generic wrapper is useless, in the sense that it describes every single element of an HTML page. More specialization is required, particularly in the HTMLElement concept. Towards this, we exploit the highly structured and information rich HTML element description provided by modern browsers. Such information includes, among others, the text contained inside the element, its attributes, the parent element under the DOM perspective, its tag name, etc. Besides this information, which is directly accessed, we also exploit calculated information that is derived if someone considers the neighborhood of some element. Such information includes, for example, the sibling order of this element being a child of its parent element and the total number of siblings. With this information in hand, a complex HTML element description can be created in CG form. Such a CG is presented in Figure 5. Note that, for clarity, Figure 5 presents a simplification (CG operation) of six CGs over the common [HTMLElement] concept presented on the left. Moreover, for space economy, a reduced version is presented, since the actual description is quite more complex.

```
[HTMLElement] ← (parent) ← [HTMLElement]
              ← (innerText) ← [Text]
              ← (tag) ← [HTMLTag]
              ← (siblingCount) ← [Integer]
              ← (siblingOrder) ← [Integer]
              ← (attribute) ← [Attribute]
```

Figure 5: An HTML element in CG form (simplified and reduced version)

We demonstrate how our generic wrapper can be specialized using the classical problem of extracting information from an electronic flea market. Figure 6 presents a snippet from a web page of such a site. Information is organized in an HTML table, where the first row holds the headers and the rest of the rows correspond to records describing offered products. We assume that we want to extract the names of the products offered.



name	user	closes in ▲	offers	price
MODEM MOTOROLA SM56 E...	WEASEL(140)	2 hours, 50 minutes	1	€ 15.00 (5112 dra.)
DIAMOND SUPRA v92 inte...	ABBOT(240)	1 μέρα, 10 hours	-	€ 22.00 (7497 dra.)
USB MICRONET MODEL 300...	hawk21(178)	2 days	-	€ 42.00 (14312 dra.)
ΣΕΙΡΙΑΚΟ MODEM 56K MIC...	✓zep(112)	2 days	-	€ 32.00 (10904 dra.)

Figure 6: A snippet from an on-line flea market

```
[Wrapper: fleaName]  ← (targetURL) ← [URL: "www.fleamarket.gr"]
                     ← (output) ← [Info]
                     ← (container) ← [HTMLElement: #16]   ← (parent) ← [HTMLElement:#15]
                                                          ← (innerText) ← [Text: "MODEM MOTOROLA SM56 E..."]
                                                          ← (tag) ← [HTMLTag: "TD"]
                                                          ← (siblingCount) ← [Integer: 5]
                                                          ← (siblingOrder) ← [Integer: 1]
                                                          ← (attribute) ← [BGCOLOR: "#FFFFFF"]
```

Figure 7: First training instance of a CG wrapper

In a real situation, where the user is not expected to be an HTML expert, the wrapper creation program should allow the identification of instances of the desired information, by simply pointing it with the mouse (we have developed such a tool which is presented in a following section). Let's say the user points to the table cell containing the name of the first product. This specializes the generic wrapper description, which takes the form presented in Figure 7.

```
[Wrapper: fleaName]  ← (targetURL) ← [URL: www.fleamarket.gr]
                     ← (output) ← [Info]
                     ← (container) ← [HTMLElement: #26]  ← (parent) ← [HTMLElement: #25]
                                                         ← (innerText) ← [Text: "DIAMOND SUPRA v92 inte..."]
                                                         ← (tag) ← [HTMLTag: "TD"]
                                                         ← (siblingCount) ← [Integer: 5]
                                                         ← (siblingOrder) ← [Integer: 1]
                                                         ← (attribute) ← [BGCOLOR: "#CCCCCC"]
```

Figure 8: Second training instance of a CG wrapper

Unfortunately, this specialized version is not general enough since it is able to extract only the training instance. A second training instance should be used, say the cell containing the name of the second product. This results in the wrapper instance presented in Figure 8.

```
[Wrapper: fleaName]  ← (targetURL) ← [URL: www.fleamarket.gr]
                     ← (output) ← [Info: X]
                     ← (container) ← [HTMLElement]  ← (parent) ← [HTMLElement]
                                                    ← (innerText) ← [Text: ?X]
                                                    ← (tag) ← [HTMLTag: "TD"]
                                                    ← (siblingCount) ← [Integer: 5]
                                                    ← (siblingOrder) ← [Integer: 1]
                                                    ← (attribute) ← [BGCOLOR]
```

Figure 9: Generalization result of CG wrapper instances of Figure 7 and Figure 8

Using the generalization operation of the CG theory for the two CG wrapper instances, a generic wrapper describing (extracting) both product names can be created (Figure 9). This wrapper is generic enough to extract all product names of the table in Figure 6, but it also extracts the first header cell. Further specialization of our CG wrapper is required to exclude the header cell. This can be established over the HTML element that is the parent of the element containing the extracted information. This element refers to a row of the product table. Excluding this row is as simple as requesting that this element's sibling order is greater than one. The final wrapper is presented in Figure 10. Note that the concept of the CG wrapper that contains the desired information ([Info]) is fed by the [Text: ?X] concept, since this part of the web page contains the desired information. In addition, parts of the final wrapper description that do not affect the accuracy of the wrapper, such as the [BGCOLOR] can be dropped out.

Finally, regular expressions can be used over the initially extracted information in order to fine-tune the output. For example, extracting the price in euros from the flea market example, requires the replacement of ?X with some proper regular expression that is applied over X.

Thus, training a CG-Wrapper, is a set of automatic generalization and manual specialization tasks that results in a model (CG) that accurately describes the desired information inside a web page.

```
[Wrapper: fleaName]   ← (targetURL) ← [URL: www.fleamarket.gr]
                      ← (output) ← [Info: X]
                      ← (container) ← [HTMLElement]   ← (parent) ← [HTMLElement] ← (siblingOrder) ← [Integer: >1]
                                                      ← (innerText) ← [Text: ?X]
                                                      ← (tag) ← [HTMLTag: "TD"]
                                                      ← (siblingCount) ← [Integer: 5]
                                                      ← (siblingOrder) ← [Integer: 1]
```

Figure 10: The final CG wrapper modeling the product names of the table in Figure 6

We propose two execution models for our CG-Wrappers, a naive and an optimized one. According to the *naive execution model*, we iterate over all the nodes of the HTML tree trying to satisfy the constraints imposed by the wrapper components. In the *optimized execution model* we first do some short of filtering, to exclude nodes that are definitely irrelevant. For example, the wrapper of Figure 10 can be evaluated only over the nodes that have a TD tag. Selecting only those nodes is possible by exploiting the browser's application programming interface (API). The semantics of both execution models are derived from the CG theory: The evaluation of a CG-Wrapper is the result πv of a projection operation that projects the container part u of the wrapper over an HTML node v expressed as CG.

```
[HTMLElement: #25]   ← (siblingOrder) ← [Integer: 3]

[HTMLElement: #26]   ← (parent) ← [HTMLElement: #25]
                     ← (innerText) ← [Text: "DIAMOND SUPRA v92 inte..."]
                     ← (tag) ← [HTMLTag: "TD"]
                     ← (siblingCount) ← [Integer: 5]
                     ← (siblingOrder) ← [Integer: 1]
                     ← (attribute) ← [BGCOLOR: "#CCCCCC"]
```

Figure 11: Two nodes of an HTML tree, in CG form (partially presented)

For example, consider the two CGs of Figure 11 which refer to the table of Figure 6, representing the second product row and the first cell of this row, respectively. Projecting the container part of the CG wrapper of Figure 10 over the second CG of Figure 11 results in an instantiated CG wrapper where the unbound X referent of the [Text: ?X] concept have been unified with "DIAMOND SUPRA v92 inte...". Note that, the exact projection involves also a replacement of the concept [HTMLElement: #25] of the second CG, with the CG definition of this concept (that is, the first CG in Figure 11). This inner task corresponds to the expansion operation of the CG theory, where a concept is replaced by its CG definition. The final instantiated wrapper is displayed in Figure 12.

```
[Wrapper: fleaName]   ← (targetURL) ← [URL: www.fleamarket.gr]
                      ← (output) ← [Info: "DIAMOND SUPRA v92 inte..."]
                      ← (container) ← [HTMLElement]   ← (parent) ← [HTMLElement] ← (siblingOrder) ← [Integer: >1]
                                                      ← (innerText) ← [Text: "DIAMOND SUPRA v92 inte..."]
                                                      ← (tag) ← [HTMLTag: "TD"]
                                                      ← (siblingCount) ← [Integer: 5]
                                                      ← (siblingOrder) ← [Integer: 1]
```

Figure 12: The wrapper of Figure 10 after applying it over the CGs of Figure 11

3.3 Reusing CG-Wrappers

The CG-Wrap model, is expressive enough to handle nested wrapper definitions, that is, wrappers that are defined in terms of other wrappers. Such a very useful case, is the definition of a *looping wrapper* that collects results from chained pages containing search results. Consider for example the typical case in which an on-line store presents the results of some user query in individual pages containing 10 items each. In such cases, at the bottom of all pages but the last one, there is a link to the next result page, usually named "Next Page". A looping wrapper is capable of extracting information from all results pages by automatically following the "Next Page" link.

Thus, a *looping CG-Wrapper* (Figure 13) is a combination of a data collector wrapper and a loop definer wrapper. A *data collector* (Wrapper:#1) is a typical CG-Wrapper that extracts information from a web page. A *loop definer* (Wrapper:#2) is a CG-Wrapper that extracts the URL of the next page, in the case of information that is presented in a sequence of pages. These two wrappers have a common target URL.

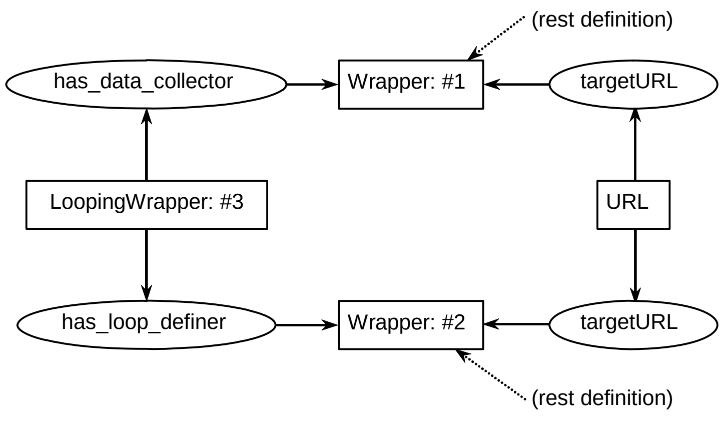


Figure 13: A looping wrapper in CG notation.

The evaluation of a looping wrapper is presented in Figure 14. First, the data collector is executed and the extracted information is appended to the already extracted results. Then, the execution of the loop definer wrapper follows which extracts from the same page the URL of the "Next Page" link. If this second wrapper brings results then the target URL of the data collector is updated. These steps are repeated until the loop definer fails to extract information.

```
LoopingWrapperExecutor(LoopingWrapper:#3)
begin
    Results:=∅;
    repeat
        WrapperExecutor(Wrapper:#1, subResults);
        Results:=AggregateResults(Results, subResults);
        WrapperExecutor(Wrapper:#2, nextURL);
        UpdateWrapper(Wrapper:#1, URL, nextURL);
    until nextURL=null;
end;
```

Figure 14: Abstract execution model of a looping wrapper.

4 Comparison Chart Building with CG-Wrappers

In this section, we identify problems involved in the comparison chart building task and propose visual and ontology driven approaches that can provide substantial automation to the whole task.

4.1 Locating Product Specification Pages

Building a comparison chart for a certain type of products using information presented in web pages requires, first of all, to locate those pages. Without doubt, the web sites of the various brands is the best place to visit. Locating such sites on the web is a relatively simple task. All that someone has to do is to either try some "URL guessing" heuristics using the www.<brand>.com pattern for known brands or use a search engine (or a portal) to locate an e-shop selling the desired category of products, where all major brands are usually mentioned. Having a brand's URL makes the product specification page detection a couple of clicks task. From a brand's main page someone has to follow the "Products" link to go to a page where a complete list of links to various products is available. It is remarkable how strong the above heuristic is. The detailed specifications of a particular product are usually displayed either inside the product's page or in a separate, dedicated page accessible from the product's main page. The above organizations are depicted in Figure 15. It is clear that, even considering that the URLs to brand sites are known, some automation is required towards collecting all the URLs to product specification pages.

```
Brand's Main Page                        | Brand's Main Page
    Product List Page                    |     Product List Page
        Specific Product Page (with specs)|         Specific Product Page
                                         |             Product's Specification Page
```

Figure 15: Typical location of a product's specification information in a brand site

We have developed a URL wizard that allows the average user to visually manipulate a web page and collect information presented in it. For the purpose of collecting URLs where products are presented, the user can exploit the product list page of a brand site where links to all available products are provided. He/she simply points (or selects) the anchor object(s) inside such a page and asks for URL harvesting from a context menu. For better manipulation, our tool provides a tree view of the web page as well. This tree view is synchronized with the browser window (see Figure 19 in Section 5), that is, when the user points over a page element in the browser window, the corresponding branch in the tree representation is automatically highlighted and vice-versa.

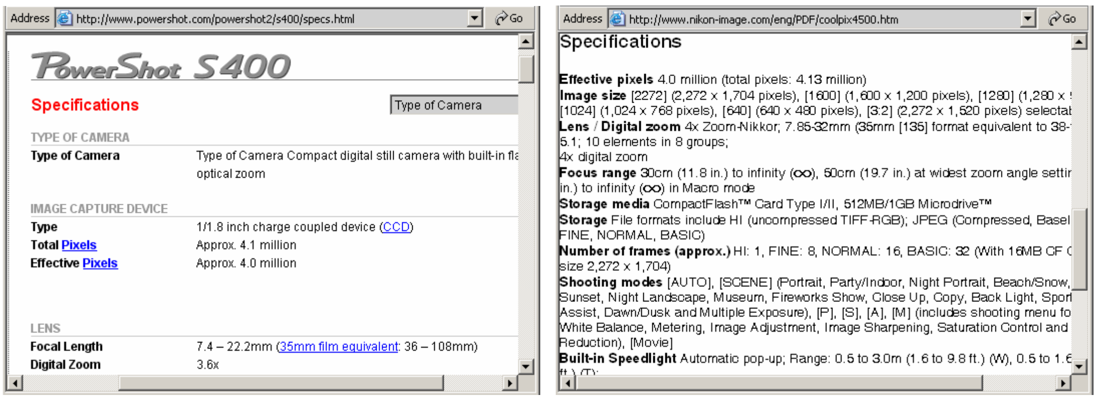
The above approach works perfectly for sites following the organization presented in Figure 15 (left). When the product has a dedicated specification page we train a CG wrapper that learns how to find the anchor to the specification page, inside the product's main page.

4.2 Collecting and Merging Specification Information

The main difficulty in comparison chart building stems mainly from the fact that, product features are not presented in a uniform way inside specification pages. Figure 16 demonstrates how diverse two specification pages could be, although they both refer to similar products (digital cameras). Not only the layout of the pages is different, which renders most of the HTML tag based information extraction methods obsolete, but the exact vocabulary used across brands also varies. The latter, makes the regular expression based extraction troublesome, as well. There exist though two strong, "per brand" regularities that seem worth to exploit:

- information in specification pages is usually presented in feature-value pairs enclosed in adjacent HTML tags, and
- the vocabulary used by each brand to refer to product features is almost fixed.

The above two regularities suggests that a dual approach is required: *first locate the feature, then locate and extract the nearby value*. Since this combination works at the brand level, the final obstacle is to integrate the "per brand" partial results under a common schema.



http://www.powershot.com/powershot2/s400/specs.html

Specifications

TYPE OF CAMERA

Type of Camera	Type of Camera Compact digital still camera with built-in fla optical zoom

IMAGE CAPTURE DEVICE

Type	1/1.8 inch charge coupled device (CCD)
Total Pixels	Approx. 4.1 million
Effective Pixels	Approx. 4.0 million

LENS

Focal Length	7.4 – 22.2mm (35mm film equivalent: 36 – 108mm)
Digital Zoom	3.6x

http://www.nikon-image.com/eng/PDF/coolpix4500.htm

Specifications

Effective pixels 4.0 million (total pixels: 4.13 million)
Image size [2272] (2,272 x 1,704 pixels), [1600] (1,600 x 1,200 pixels), [1280] (1,280 x
[1024] (1,024 x 768 pixels), [640] (640 x 480 pixels), [3:2] (2,272 x 1,520 pixels) selectab
Lens / Digital zoom 4x Zoom-Nikkor; 7.85-32mm (35mm [135] format equivalent to 38-
5.1; 10 elements in 8 groups;
4x digital zoom
Focus range 30cm (11.8 in.) to infinity (∞), 50cm (19.7 in.) at widest zoom angle settir
in.) to infinity (∞) in Macro mode
Storage media CompactFlash™ Card Type I/II, 512MB/1GB Microdrive™
Storage File formats include HI (uncompressed TIFF-RGB); JPEG (Compressed, Basel
FINE, NORMAL, BASIC)
Number of frames (approx.) HI: 1, FINE: 8, NORMAL: 16, BASIC: 32 (With 16MB CF C
size 2,272 x 1,704)
Shooting modes [AUTO], [SCENE] (Portrait, Party/Indoor, Night Portrait, Beach/Snow,
Sunset, Night Landscape, Museum, Fireworks Show, Close Up, Copy, Back Light, Sport
Assist, Dawn/Dusk and Multiple Exposure), [P], [S], [A], [M] (includes shooting menu fo
White Balance, Metering, Image Adjustment, Image Sharpening, Saturation Control and
Reduction), [Movie]
Built-in Speedlight Automatic pop-up; Range: 0.5 to 3.0m (1.6 to 9.8 ft.) (W), 0.5 to 1.6
ft.) (T);

Figure 16: Same type of products but diversity in the way specification information is presented

We have selected to use a product ontology as a common schema. As the semantic web evolves, ontologies describing products of any kind are expected to become available. Such ontologies can be used to map features expressed in a brand's vocabulary to ontology elements.

CGs are a proper candidate for describing ontological information. They offer a unified and simple representation formalism that covers a wide range of other data and knowledge modelling formalisms and allow matching, transformation, unification and inference operators to process the knowledge that they describe [23]. Having already used CGs to model and train our CG-Wrappers, CG based ontological knowledge can be easily incorporated and contribute towards knowledge-based wrappers.

Consider, for example, two wrappers that extract the focal length and the optical zoom from specification pages of digital cameras. Background knowledge regarding the relation that exists between optical zoom and focal length can be used to modify the kind of information that the focal length wrapper is expected to locate, assuming that the optical zoom wrapper has already extract information. In another case, having selected the brand of a processor, should automatically prevent the extraction of information for certain, incompatible, motherboard models.

Although ontological knowledge is expected to become available in RDF/RDFS, the semantic web's language, converting this encoding to CGs is not an issue ([17], [6]). Furthermore, CGs provide a "ready to use" framework for reasoning. This is not the case, at the moment, for RDF/RDFS.

As a result, we propose a dual wrapper approach for extracting feature-value pairs from product specification pages:

- Associate a wrapper to some product feature, as it is defined in the product ontology and train the wrapper to locate that feature based on the term used by a brand.
- Use a second wrapper to extract the value of the feature.

This dual wrapper approach is justified by the fact that feature-value information is always located in adjacent HTML elements inside a web page. We can easily encode this information in our wrapper pair reducing in that way the search space of the second wrapper. Furthermore, the second wrapper becomes capable of performing a "blind" extraction in case the value of some feature is presented in an unknown way. In a "blind" extraction the wrapper extracts all the text inside some HTML tag, because "it knows" that the information is there. This is obviously better than an exact-or-nothing approach. In addition, we are not depended on absolute positioning to refer to HTML nodes but we follow a "relative to textual information" methodology instead which is more robust to small page changes. This is very crucial, since many commercial sites tend to make frequent alterations to their sites to prevent wrapping. The same holds for the advertisement banners and special offers, the frequent addition and removal of which, turn obsolete wrappers that use absolute positioning.

Figure 17 displays a dual wrapper ([DualWrapper: CanonDigitalZoom]) extracting feature-value information (digital zoom of a digital camera model). It is defined in terms of a feature locator wrapper ([Wrapper: #1]) that locates the table cell ([HTMLTag: "TD"]) containing the text "Digital Zoom", and a value extractor wrapper ([Wrapper: #2]) that extract the value of the feature. The second wrapper is modelled to search in the table cell that is right after the cell the first wrapped worked with. This correlation is established over the parameter ?X.

The dual wrapper of Figure 17 can be used as a data collector in the comparison chart building problem, in the way the simple CG-Wrapper was used in the flea market problem (Section 3.2).

```
[Wrapper: #1] ← (targetURL) ← [URL: www.powershot.com/powershot2/s400/specs.html]
              ← (output) ← [Info: X]
              ← (container) ← [HTMLElement]  ← (innerText) ← [Text: "Digital Zoom"]
                                             ← (tag) ← [HTMLTag: "TD"]
                                             ← (siblingOrder) ← [Integer: ?X]

[Wrapper: #2] ← (targetURL) ← [URL: www.powershot.com/powershot2/s400/specs.html]
              ← (output) ← [Info: V]
              ← (container) ← [HTMLElement]  ← (innerText) ← [Text: ?V]
                                             ← (tag) ← [HTMLTag: "TD"]
                                             ← (siblingOrder) ← [Integer: X+1]

[DualWrapper: CanonDigitalZoom]  ← (featureWrapper) ← [Wrapper: feature]
                                 ← (valueWrapper) ← [Wrapper: value] ← (output) ← [Info: ?V]
                                 ← (output) ← [Info: V] ← (has_value) ← [Digital Zoom]
```

Figure 17: A dual wrapper extracting digital zoom information

5 A Framework for Information Extraction with CG-Wrappers

In this section, we describe the system architecture of Aggregator, a comparison chart builder that implements the ideas discussed in the previous sections. In addition we present a small scale, demonstrational usage, using a prototype implementation.

5.1 System Architecture

The Aggregator is a tool aiming at helping the user to rapidly create side-by-side comparison charts using product specification web pages. It consists of four main modules (Figure 18):

- the interactive wrapper creator,
- the evaluator,
- the knowledge based module, and
- the publisher

The *interactive wrapper creator* is a sophisticated visual environment that allows the user to train wrappers. It consists of a *web browser* instance accompanied by the *DOM tree component* and interconnected in such a way that allows the user to focus on the elements of a web page by simply using the mouse (Figure 19). This is established with the extensive use of an *HTML parser* that gives access to all the elements of a web page. Finally, this module includes a *product feature list* which can be either derived by a predefined product ontology or manually edited by the user. The wrapper creator module allows the user to navigate to desired web locations, where product specification information is presented, and visually map page elements to feature-value pairs of a corresponding wrapper template.

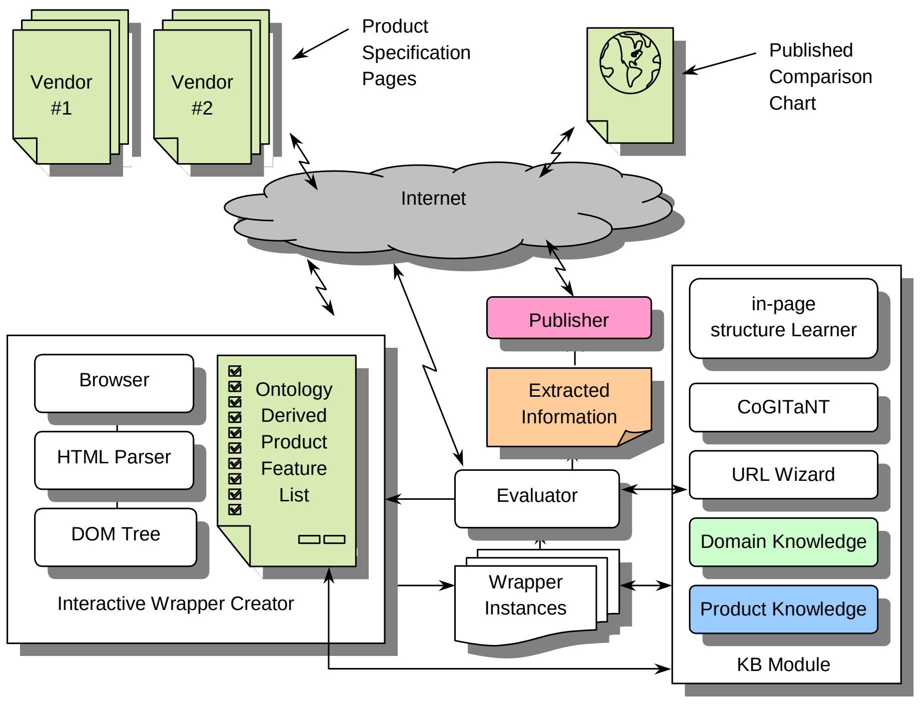


Figure 18: System Architecture

The *evaluator module* "runs" the created wrappers and actually does the information extraction. The extracted information can be published on the web by the *publisher module* in the form of a static web page. In addition, it is possible to save it as a spreadsheet table.

The *knowledge based* (KB) module is basically a conceptual graph inference engine (the core of which has been developed in our past work in [25] and [24]). Its main component is *CoGITaNT* ([10],[22]), a library of C++ classes allowing the development of applications based on the CGs. CoGITaNT allows the handling of CGs using an object oriented approach and offers a great number of functionalities on them such as creation, modification, projection, definition of rules, inputs/outputs, etc. Furthermore, CoGITaNT can be extended since it provides the programming interface to define new operations, like for example, customized concept and relation matching operations and rule execution methods. The knowledge included in the KB module is divided into *domain knowledge* and *product knowledge*. The former, is mostly related to the DOM specification and includes concept types related to the DOM elements and relation types that allow us to describe the various usage constraints between DOM elements. The *product knowledge*, which is also encoded in CGs, serves in three ways:

- defines the potential features/attributes for which we may build wrappers, in a form of a product ontology,
- provides generic wrapper templates which the user should make more detailed, and
- gives insight for the values that a particular wrapper should search for.

The presence of product knowledge is optional since Aggregator can operate without this information but at the cost of reduced precision in the extracted information.

The *URL Wizard* is an important sub-component of the KB module that helps the user to quickly populate the list of URLs that will be the target of the various wrappers. This is done using minor user input, mainly in the form of link traversal tracking. Internally, this module uses a proper, predefined CG-Wrapper.

Finally, the *in-page structure learner* is responsible to determine how the feature-value pairs are organized inside a product specification page. This is done by means of a generalization operation as soon as two wrappers have been visually trained by the user. The learned pattern is used to partially detail the remaining wrapper templates. This will reduce the user effort for wrapper training since the system is becoming able to suggest possible page element for wrapper part assignments.

The prototype of Aggregator runs on wintel machines. The user interface is built in Delphi (Figure 19) and makes extensive use of the Microsoft's HTML parser (which is used in Internet Explorer). The knowledge based components are built in C++ and make use of the CoGITaNT library.

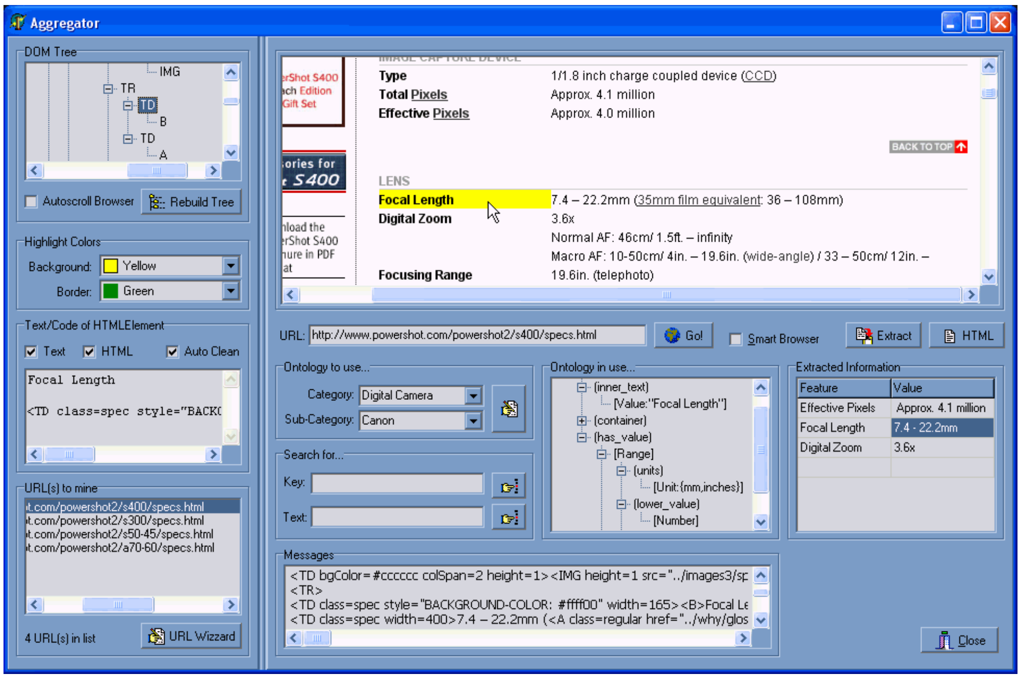
5.2 Case Study

We have done a small scale evaluation study of Aggregator. We asked four experienced web users to create a feature/value comparison chart for the digital cameras of two brands. Two of the users (1st group) used the Aggregator agent while the rest (2nd group) used a web browser and a spreadsheet application. All users were provided with two URLs, one for each brand site, which were the entry pages leading to individual product pages. Regarding the individual product pages, both sites had the typical organization presented in Figure 15 (right), that is, the products' page was giving access to the pages of individual products from where access was provided to the specification page of a particular product. None of the users was aware of this organization.

In addition, we defined which were the exact features of interest and provided all users with the proper product feature list. The features of interest were: *model name*, *CCD resolution*, *focal length*, *optical zoom*, *digital zoom*, *shutter speed*, *white balance*, *flash modes*, *storage media* and *power source*. Exact value extraction was requested only for *CCD resolution*, *focal length*, *optical zoom*, *digital zoom* and *shutter speed*.

The first user group used the URL wizard to train Aggregator how to locate the individual product pages. Starting from the given Brand#1 central page, the users of the first group used the visual tools of the Aggregator to quickly collect the URLs of all the product pages (*ProductURLs*). Just moving around the mouse, both users were able to rapidly locate the page elements (two HTML tables) that contained all the anchors to the individual product pages and ordered Aggregator (from a context menu) to record those URLs. Then, they recorded a navigation pattern from a product's main page to the product's specification page. This resulted in a wrapper that given the initial product URL list produced a list with the URLs of the specification pages (*SpecURLs*). The same task was repeated for the second brand site.

After the target pages for information extraction had been defined, each user of the 1st user group had to train the "dual wrappers" that would perform the actual information extraction. With a product specification page loaded into the embedded web browser and a predefined digital camera ontology available, the users had to select the features they were told from the digital camera ontology. The system then, internally, created the corresponding dual wrapper templates, presented the first one to the user and waited from him/her to visually associate an element of the specification page (for example, a table cell) with a wrapper element (Figure 19). After that, both users had to point over the page element that contained the value of the attribute under consideration. These two steps are enough for the Aggregator to create a wrapper to handle this specific attribute-value pair. The generated wrapper can be immediately evaluated over the *SpecURLs* list. The task is repeated for a second wrapper. These two wrapper instances allow the system to automatically determine the repetitive HTML structures used in the specification page to present the attribute-value pairs. We remind here that this is done with a generalization operation between the two user defined wrappers.



Background: 🟨 Yellow
Border: 🟩 Green

[✓] Text [✓] HTML [✓] Auto Clean

```
Focal Length

<TD class=spec style="BACKG
```

t.com/powershot2/s400/specs.html
t.com/powershot2/s300/specs.html
t.com/powershot2/s50-45/specs.html
t.com/powershot2/a70-60/specs.html

4 URL(s) in list

Type	1/1.8 inch charge coupled device (CCD)
Total Pixels	Approx. 4.1 million
Effective Pixels	Approx. 4.0 million
LENS	
Focal Length	7.4 – 22.2mm (35mm film equivalent: 36 – 108mm)
Digital Zoom	3.6x
Focusing Range	Normal AF: 46cm/ 1.5ft. – infinity Macro AF: 10-50cm/ 4in. – 19.6in. (wide-angle) / 33 – 50cm/ 12in. – 19.6in. (telephoto)

URL: http://www.powershot.com/powershot2/s400/specs.html

Category: Digital Camera
Sub-Category: Canon

Key:
Text:

```
(inner_text)
  [Value:"Focal Length"]
(container)
(has_value)
  [Range]
    (units)
      [Unit:{mm,inches}]
    (lower_value)
      [Number]
```

Feature	Value
Effective Pixels	Approx. 4.1 million
Focal Length	7.4 - 22.2mm
Digital Zoom	3.6x

```
<TD bgColor=#cccccc colSpan=2 height=1><IMG height=1 src="../images3/sp
<TR>
<TD class=spec style="BACKGROUND-COLOR: #ffff00" width=165><B>Focal Le
<TD class=spec width=400>7.4 – 22.2mm (<A class=regular href="../why/glos
```

Figure 19: The wrapper training/evaluation screen of Aggregator.

It is worth mentioned here that Brand#1 had no visible textual model name information. Instead, it provided the model name in a form of a picture. That was no problem for the users of the 1st group since they assigned this picture's ALT property as the value of the model name feature. To the contrary, the users of the 2nd group had to manually type the model name in a spreadsheet cell.

For Brand#1, the system was able to detail automatically the seven out of the eight remaining wrappers. The missing case was related to the Optical Zoom feature because this information was included inside the general description of the product rather than in a dedicated feature-value pair. As a result this case required the user to manually train the corresponding wrapper. This issue, demonstrates the advantage of searching for both feature and value related page elements, instead of just value elements. Although this particular wrapper was about "Optical Zoom", it's feature part was related by the user to a page element with information about "Type of Camera". By focusing on a tiny part of an HTML page, it is possible to apply more computationally complex methods to extract an exact value for some feature.

A total of 20 wrapper instances was created for both Brand#1 and Brand#2 sites (10 features times the number of brand sites). The time required to perform this information extraction task is presented in Table 1. Although the recall factor was 100% for both brands, that is, all the desired features were located inside the product pages, the precision factor was 70% for Brand#1 and 50% for Brand#2. These precision numbers are not discouraging because although Aggregator failed to extract exact values for certain features, it had extracted a bigger portion of information that included the exact information. This, of course, prevents the user to query the complete resulted comparison chart in an SQL fashion, but does not prevent him/her to manually examine the chart and make an informed purchase decision.

Table 1: Case study time results

		1st group (using Aggregator)			2nd group (using browser and spreadsheet)		
		user1	user2	average time	user 3	user 4	average time
brand #1 (8 products)	training	254 sec	292 sec	273 sec			
	extraction*	18 sec	18 sec	18 sec	8x199 sec	8x183 sec	8x191 sec
	total	272 sec	310 sec	291 sec	1592 sec	1464 sec	1528 sec
brand #2 (6 products)	training	320 sec	328 sec	324 sec			
	extraction*	12 sec	12 sec	12 sec	6x240 sec	6x224 sec	6x232 sec
	total	332 sec	340 sec	336 sec	1440 sec	1344 sec	1392 sec
Complete Task		604 sec	650 sec	*627 sec*	3032 sec	2808 sec	*2920 sec*
		per page average extraction time		45 sec	per page average extraction time		209 sec

* extraction times for the 2nd group are given in terms of the average time required to extract values from a single product specification page

It is worth mentioning that, although it takes more time for an Aggregator user to train the wrappers for a single brand page than it takes another user to manually extract (with copy-paste) the same information from the same page into a spreadsheet, additional product specification pages of the same brand are processed rapidly, resulting in a lower per page average extraction time (45 versus 209).

6 Conclusions and Future Work

Product specification pages provided on-line at various brand sites, are an excellent source of information to automatically create side-by-side comparison charts for "informed" e-shopping. Apart from the information rich nature of such pages, they also use an in-site fixed vocabulary to refer to the various features of the advertised products and present these features using repetitive HTML tag combinations of arbitrary complexity.

In this chapter, we have proposed a knowledge based approach on the comparison chart building problem. Our method is two fold: First, we exploit vertical (in-page) similarities, that is, similarities in the way features are presented inside a product specification page. We visually identify feature-value information, map the surrounding HTML tags to predefined generic wrappers expressed as Conceptual Graphs and use the generalization operation to "learn" how information is presented inside a specification page of a brand site. This way, additional features can be located and the related values can be extracted automatically, although sometimes at a low precision ratio because the desired information is mixed with some extra text. In addition we exploit horizontal (in-site) similarities, that is, similarities across different product specification pages of the same brand. These are vocabulary and page layout similarities.

Furthermore, we argue that a product ontology and product background knowledge can speed up the wrapper training process and improve the precision ratio of the extracted information. We have proposed the use of the Conceptual Graph knowledge representation and reasoning formalism for the knowledge based part of our approach, mainly due to their expressiveness power and the analogy between operations provided by the CG theory and operations required to train and apply a wrapper. In addition, CGs allow to easily integrate ontological knowledge about the product type under consideration. This feature can contribute to the resiliency and adaptivity of our approach beyond the scope of [20], by adding rules and axiomatic knowledge that can alter the way wrappers are described under certain conditions that hold on other wrappers or the data they extracted.

Finally, we have outlined the Aggregator, a side-by-side comparison chart builder that is based on the above techniques and provides visual tools to make the whole task easier.

Much more work is required, mainly in the ontology utilization part of our approach. We firstly aim at providing automatic utilization of on-line ontologies expressed in XML/RDF, in the way we utilize metadata information in [25] and [24]. We also plan to use Aggregator for side-by-side comparison of learning objects which have XML expressed metadata and for which we have already proposed knowledge based approaches based on CGs ([25], [24]).

Additionally, more work is required in the value extraction part of our method. Exact value extraction will require extensive use of regular expressions and probably of NLP techniques, but will allow us to query more fields of the resulted comparison chart in an SQL fashion. The fact that the part of a page that contains the exact value of a feature can be isolated and the kind of the expected value can be defined in the product type ontology, suggests that the whole problem is tractable at a good extend.

Finally we aim at improving the adaptability of our approach by creating brand-independent wrappers. From some early attempts, this is already possible for feature-value pairs that are crucial features of a product, like for example, the frequency of a processor or the screen diagonal dimension of a TV set. Apart from having relatively simple values, such features are usually presented alone inside a page, because are strong purchase decision criteria.

7 References

[1] Adelberg B. "NoDoSE: A Tool for Semi-Automatically Extracting Structured and Semi-Structured Data from Text Documents", *SIGMOD Record,* 27(2), pp. 283-294, 1998.

[2] Ashish N. and Knoblock C. "Wrapper Generation for Semi-structured Internet Sources". In *Proceedings of Workshop on Management of Semi-structured Data*, 1997.

[3] Baumgartner R., Flesca S. and Gottlob G. "Declarative Information Extraction, Web Crawling and Recursive Wrapping with Lixto". In *Proceedings of the 6th In-*

ternational Conference on Logic Programming and Non-monotonic Reasoning, Springer-Verlang, LNCS 2173, 2001.

[4] Baumgartner R., Flesca S. and Gottlob G. "Visual Web Information Extraction with Lixto". In *Proceedings of the 27th International Conference on Very Large Data Bases*, pp.119–128, 2001.

[5] Baumgartner R., Gottlob G. and Herzog M. "Visual Programming of Web Data Aggregation Applications", In *on-line proceedings of IJCAI'03 workshop on Information Integration on the Web (IIWeb-03)*, http://www.isi.edu/info-agents/work-shops/ijcai03/papers/Herzog-ijcai03-herzog.pdf, 2003.

[6] Berners-Lee T. "Conceptual Graphs and the Semantic Web", on-line document, http://www.w3.org/ DesignIssues/CG.html

[7] Berners-Lee T., Hendler J. and Lassila O. "The Semantic Web", *Scientific American*, May 2001.

[8] Buttler D., Liu L. and Pu C. "A Fully Automated Object Extraction System for the World Wide Web". In *Proceedings of the 21th International Conference on Distributed Computing Systems*, pp.361–370, 2001.

[9] Chidlovskii B. "Wrapper generation by k-reversible grammar induction". In *Proceedings of the Workshop on Machine Learning and Information Extraction*, Berlin, Germany, 2000.

[10] CoGITaNT library, available under GPL at: http://cogitant.sourceforge.net

[11] Cohen W. W. and Fan W. "Learning page-independent heuristics for extracting data from web pages". In *Proceedings of the Eighth International World Wide Web Conference (WWW-99)*, Toronto, 1999.

[12] Cohen W. W. and Jensen L. S. "A Structured Wrapper Induction System for Extracting Information from Semi-structured Documents". In *Proceedings of IJCAI 2001Workshop on Adaptive Text Extraction and Mining*, 2001.

[13] Cohen W. W. "Recognizing structure in web pages using similarity queries". In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, 1999.

[14] Conceptual Graphs Standard Working Draft, http://www.jfsowa.com/cg/cgstand.htm

[15] Corbett D., "A Method for Reasoning with Ontologies Represented as Conceptual Graphs", In *M. Brooks, D. Corbett and M. Stumptner (Eds.): AI 2001*, Springer Verlag, LNAI 2256, pp.130-141, 2001.

[16] Crescenzi V., Mecca G. and Merialdo P. "RoadRunner: Towards Automatic Data Extraction from Large Web Sites", In *Proceedings of the 26th International Conference on Very Large Database Systems*, pp.109-118, 2001.

[17] Delteil A., Faron-Zucker C. and Dieng R. "Extension of RDFS based on the CGs Formalisms". In *Proceedings of the ICCS 2001*, LNAI 2120, Springer Verlag, pp.275-289, 2001.

[18] Document Object Model (DOM), http://www.w3.org/DOM/

[19] Doorenbos R. B., Etzioni O. and Weld D. S. "A scalable Comparison Shopping Agent for the World Wide Web", In *Proceedings of the 1st International Conference on Autonomous Agents*, 1997.

[20] Embley D. W., Campbell D. M., Jiang Y. S., Liddle S. W., Ng Y.-K., Quass D. and Smith R. D. "A Conceptual-Modelling Approach to Extracting Data from the Web". In *Proceedings of International Conference on Conceptual Modeling / the Entity Relationship Approach*, pp.78-91, 1998.

[21] Freitag D. and Kushmerick N. "Boosted Wrapper Induction". In *Proceedings of the 17th National Conference on Artificial Intelligence*, pp.577-583, 2000.

[22] Genest D. and Salvat E. "A Platform Allowing Typed Nested Graphs: How CoGITo Became CoGITaNT", In *Proceedings of the 6th International Conference on Conceptual Structures*, Springer-Verlag, LNAI 1453, pp.154-161, 1998.

[23] Gerbe O. and Mineau G. W. "The CG Formalism as an Ontolingua for Web-Oriented Representation Languages". In *Proceedings of the ICCS 2002*, Springer Verlag, LNAI 2392, pp. 205-219, 2002.

[24] Kokkoras F. and Vlahavas I. "Metadata Aware Peer-to-Peer Agents for the e-Learner", A "Hercma03" Symposium on "AI Techniques in e-Learning", Athens, Greece, 2003 (accepted for publication).

[25] Kokkoras F., Sampson D. and Vlahavas I. "A Knowledge Based Approach on Educational Metadata Use", *Post-proceedings of the 8th Panhellenic Conference in Informatics*, Y. Manolopoulos, S. Evripidou and A. Kakas (Eds.), Springer-Verlag, LNCS 2563, 2003.

[26] Kushmerick N., Weld D. S. and Doorenbos R. B. "Wrapper Induction for Information Extraction". In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pp.729–737, 1997.

[27] Laender A. H. F., Ribeiro-Neto B. A. and da Silva A. S. "DEByE - Data Extraction by Example", *Data and Knowledge Engineering*, 40(2), pp.121-154, 2001.

[28] Laender A., Ribeiro-Neto B., da Silva A. and Teixeira J. "A Brief Survey of Web Data Extraction Tools", *SIGMOD Record,* 31(2), June 2002.

[29] Liu L., Pu C. and Han W. "XWRAP: An XML-Enabled Wrapper Construction System for Web Information Sources". In *Proceedings of the 16th IEEE International Conference on Data Engineering*, pp. 611–621, 2000.

[30] Muslea I., Minton S. and Knoblock C. "STALKER: Learning Extraction Rules for Semi-structured Web-based Information Sources". In *Proceedings of AAAI-98 Workshop on AI and Information Integration*, pp 74–81, 1998.

[31] Muslea I., Minton S. and Knoblock C. "Wrapper induction for semi structured information sources". *Journal of Autonomous Agents and Multi-Agent Systems*, 16(12), 1999.

[32] Sahuguet A. and Azavant F. "Building intelligent web applications using light-weight wrappers", *Data and Knowledge Engineering*, 36(3), pp.283-316, 2001.

[33] Sahuguet A. and Azavant F. "Building light-weight wrappers for legacy web data sources using W4F". In *Proceedings of VLDB '99,* pages pages 738-741, 1999.

[34] Sowa J. "Conceptual Structures: Information Processing in Mind and Machine". *Addison-Wesley Publishing Company*, 1984.

[35] Yamada Y., Ikeda D. and Hirokawa S. "Automatic Wrapper Generation for Multi-lingual Web Resources". In *Proceedings of the 5th International Conference on Discovery Science*, Springer-Verlag, LNCS 2534, pp.332–339, 2002.

[36] Yamada Y., Ikeda D. and Hirokawa S. "Expressive Power of Tree and String Based Wrappers", In *on-line proceedings of IJCAI'03 workshop on Information Integration on the Web (IIWeb-03)*, http://www.isi.edu/info-agents/workshops/ijcai03/papers/Herzog-ijcai03-herzog.pdf, 2003.

[37] Yamada Y., Ikeda D. and Hirokawa S. "SCOOP: A Record Extractor without Knowledge on Input". In *Proceedings of the 4th International Conference on Discovery Science*, Springer-Verlag, LNAI 2226, pp.428–487, 2001.