Learning Object Model from Product Web Pages

Shiren Ye and Tat-Seng Chua
School of Computing, National University of Singapore, Singapore 117543
{yesr |chuats }@comp.nus.edu.sg

ABSTRACT
This paper presents an automated approach to learning object model based on useful object data or product descriptions extracted from complex web pages. The common structure within similar pages from the same web site could reflect the schema hidden in the object model. First, we identify object region covering the descriptions of object data by removing irrelevant contents from the web page. Second, we partition the contents of different object data appearing in the same object region. Third, we construct the abstract object model from these data object instances extracted. The object model is equivalent to ontology in this domain and could be used to match the corresponding object data in the web site more precisely and comprehensively than the general ontology. We employ our approach to extract object models for digital cameras using the pages down-loaded from the manufacturers' web sites. Comparison with the available digital camera ontologies demonstrates that our framework is effective and efficient.

General Terms
Algorithms, Measurement, Performance

Keywords
Ontology Learning, Semantics, Information Extraction, Web Mining, Object Model

1. INTRODUCTION
With the wide spread adoption of WWW by the general public, more and more companies are beginning to manage their business and advertise their products and services on the web. It is now possible to collect, collate and compare dynamic product information from multiple web sites to support various value-added applications. An example of a typical product web page is shown in Fig. 1. We call such pages complex web pages as they contain multiple types of data components, including object region containing product descriptions, advertisement bars, product category, search and filtering panel, and navigator bar etc. Most applications only require the description of desired product without the irrelevant details. Such product information (which we call **object data**) usually cluster within a contiguous region (call **object region**) in the web page. Typically, a product web page contains only one object region, and each region lists one or more object data. To support various value-added applications, we need tools to extract and collate such product information from multiple sites, often with diverse formats and styles, and with template of unknown structure. In addition, we need an object model or ontology to provide the basis for extracting object data accurately and completely.

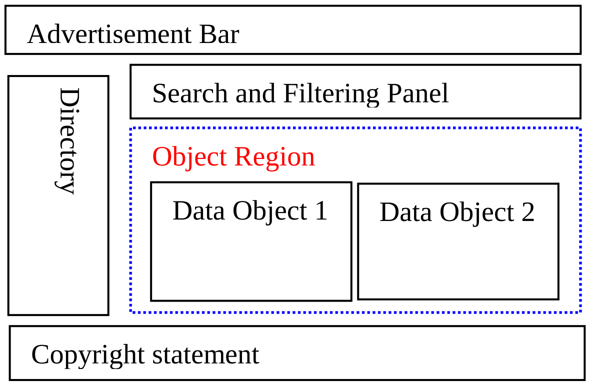


Fig. 1: The layout of a typical product description page

The main problems with extracting object model and object data from complex web pages are as follows:

a) There are many diverse sites with pages in different formats and styles. It is difficult to extract rules or patterns to handle them effectively.

b) There are many irrelevant contents intertwine with the descriptions of object data in web pages. Actually the proportion of relevant information is normally less than 10% in many product pages. The rest are irrelevant items such as the HTML tags for display formats, advertisement bar, product category, search and filtering panel, navigator bar, copyright statement, feedback form etc.

c) Most web pages depict more than one product, and thus there is more than one object instance in each object region. Although each object instance may appear to be coherent in the "formatted view" on the screen, it is actually scattered in many non-contiguous regions in raw web page. Thus one issue is how to detect the right association among object with its attribute names and values.

d) The order, number and format of attributes involved in depicting the same object data may vary greatly in different pages from different sites. For example, a page listed by a small marketing company may list only the main features of the product. However, the corresponding page owned by the manufacturer may show all descriptions of the same product (which may reach over 100 attributes). It is thus difficult to construct a fixed template or ontology to specify all desired attributes. Moreover, many new attributes and optional components might not be reflected in the training data, thus making the problem of extracting information from live web pages even more difficult.

Because of the above problems, hand-coded rules or fixed wrapper systems cannot achieve good performance on these complex web pages. Ideally, we need an automated system to extract object data (or product information) and object model. Since product pages from the same web site are normally generated by the same dynamic program or carefully maintained template, they tend to share similar structure. It is thus possible to unveil such common structure by analyzing these pages together, and use this knowledge as the basis to remove irrelevant contents while retaining relevant object information in object region.

Actually, the contents of the object data can be explored from these three aspects: (a) syntactic: that describes the rule, grammar, order and format of data representation; (b) structural: which describes the manner in which data are organized or interrelated; and (c) semantics: that gives meanings to data within a given context. In this paper, we aim to develop an unsupervised learning approach to explore the actual meaning (or semantics) of object data from the collection of stable and similar web pages. We seek to mine the object model based on the object data extracted from the original pages. The object model is the abstraction of the object data and provides the domain knowledge to guide the users and software agents on interpreting and extracting object data. It plays similar role as ontology in domain specific applications.

As the organization of complex web pages is rather complicated, it is not possible to learn the object model from the web pages directly. We thus divide the problem into three stages of: (a) identifying the object region covering product descriptions from complex web pages; (b) segmenting the object region into different object data; and (c) constructing the abstract object model based on the extracted object instances. This paper describes the details of each stage with emphasis on the last stage.

Briefly, the contents of this paper are organized as follows. Section 2 introduces related work and Section 3 presents the overall procedure for learning the object model through the detection and partitioning of object data from complex pages. Sections 4 and 5 respectively present the algorithms for detecting object region and partitioning object data. Section 6 describes the method to construct data object model. The results of our experiments and conclusions are respectively presented in Sections 7 and 8.

2. RELATED WORK

Following the intense interests in Semantic Web, research on extracting and integrating information from the web has become more and more important recently. This is because it helps to tackle the urgent business problem of collating, comparing and analyzing business information from multiple sites. Moreover, learning the hidden object model could provide the syntactic, structural and semantic descriptions of the data, and facilitates better understanding and use of data. In this respect, related works to our study include ontology learning, information extraction (IE), data integration and web page cleaning. Many researchers have focused on such problems. We briefly discuss some of them below.

Ontology learning has been becoming a hot research area in recent years [22]. It uses machine learning techniques, such as classification [13], clustering [1], inductive logic programming [11], association rules [21], concept induction [19][25], and Naive Bayes [8], to construct ontology and semantic annotation. Discovering the complicated domain ontologies, which could provide detailed description of the domain concepts from a restricted domain, is an important sub-task of ontology learning. For example, [21] employed a generalized association rule extraction algorithm to detect hierarchical relations between concepts and determine the appropriate level of abstraction at which to define relations. [30] demonstrated that the integration of machine learning techniques with knowledge acquisition from experts can both improve the accuracy of the developed domain ontology and reduce development time. However, most domain ontology acquisition approaches are still guided by human knowledge engineers, and automated learning techniques play a minor role in knowledge acquisition.

However, before object model or ontology can be built, we need to extract appropriate object data from the web pages. This is typically done using the IE tool called the wrapper [20], which is a program that extracts data from web pages and store them in a database. The wrapper could be generated either by human or learned from labeled data, both of which are labor intensive and time consuming. Moreover, the generalization of wrapper is limited when it uses the HTML tags to represent the rule. It also lacks the ability to extend to diverse sites with different representation styles.

To overcome these problems, some researches try to mine the knowledge or data in the web pages automatically [2][4][16][18] by using unsupervised learning approaches such as clustering and grammar induction, or heuristic rules. Most of these techniques require well-formatted tables, with at least two object data (records) appearing in a page. So their function is largely similar to table detection and data extraction from tables. But our study tries to handle the more complicated cases of multiple object representations in the form of table, (nested) listing and text fragment surrounded by HTML tags, with unrestricted number of object data in a page.

In a related research, web page cleaning is developed to identify redundant, useless or irrelevant components (e.g., ads bar and category) that the users are not interested in. Current research used priori knowledge or supervised learning to detect frequent templates [2], coherent content blocks [17] and site style tree [31] tied to this task. [13] used predefined unimportant tag-set and link-list to filter the irrelevant components in the pages parsed in DOM trees. However, it quite depends on the manual configure and the enumeration completeness of trivial nodes, and does not fit for the complicated pages. Here we emphasize on the development of an automated approach to detect the important portion (object region) rather than to eliminate these unimportant components in complex pages.

3. OUR APPROACHES

Although XML could offer developers more options for defining meaning in XML documents, most of current Web pages still use the traditional HTML representation. This Section gives the overview of our approach for mining the underlay meaning among the product pages. First we motivate the need for automated object model construction follow by an outline of our approach to build object model.

3.1 Automated object model construction

Ontology is a branch of philosophy that attempts to model things as they exist in the world [3]. It is particularly appropriate for modeling objects and their relationships and properties [29]. [27][28] indicated that object models and axioms are the essence of ontology. The semantic object model could therefore provide an ontology that describes the knowledge in the domain that the users are interested in. In a way, the object model encodes the syntactic, structural and semantic information about the object. It is the important component when we transform the traditional web to the Semantic Web and Web Service, whose success and proliferation depends on how quickly and cheaply we can construct domain ontologies

There are several reasons why we need automated techniques to construct object models.

a) Although many ontologies created by human experts are available in many domains, most of them cannot correctly match the real data that the users need to process. For instance, we might find an available ontology for digital camera containing 20s elements, but the pages for a kind of digital camera from a special vendor have nearly 100 detailed description items. There is a large gap between the data and the target template since their existing elements and organization is quite different. Hence, it is difficult to extract the detailed information from the source when the goal represented in another level and degree.

b) The contents of product or service pages are dynamic and constantly changing. The existing ontologies are often out-of-date and cannot cover new data descriptions. Moreover, new attributes are frequently emerging in object data. It is costly to update the object model regularly and accurately. The available classifiers, such as manual rules and learned patterns, could predict some pieces of local descriptions. However, we do not know when and which one we need to synchronize to the update descriptions, when and which one we need to extend to the new descriptions.

c) In real applications, users need to spend much time on deriving and understanding the data schema and business rules when they collect and clean information from the web. If the unsupervised learning algorithms can produce a proper object model that matches the data from the particular site well, it could reduce the cost when the users acquire the data from the web.

d) Once we have the high-quality object model reviewed by domain experts, it can be used as the basis to guide the process of object identification and extraction, and correct the errors and defaults among the object data. This helps to improve the quality of object data extraction.

e) The object model could provide the schema for database design which is used to store the extracted object data.

3.2 Outline of our approach

It is hard to learn the object model from the web pages directly. This is because the organization of complex web pages is very complicated, where irrelevant components, object names, attribute names and values about the different objects are enwind together. We thus need to apply the indirect approach by first constructing the clean and structured middle data – the object data, before learning the object model from these high-quality object instances.

One key function involved in our technique is how to evaluate the similarity among the components in the pages and through which, partition the descriptions of object data. The quality of the induced object model depends largely on the accuracy of the extracted object data. Obviously, string matching and the "bags of words" techniques cannot be used to identify irrelevant contents and partitioning object data. Although the task seems complex, we rely on the following observations to accomplish our goals.

a. In the same web site, we observe that pages about the objects in the same category, such as products, services and member listings, always have similar representation structure and similar irrelevant contents. This is because in most such cases, they are produced by the same dynamic programs or templates that are carefully maintained by the developers. Thus we could make use of the stable representation structure of the pages within a web site to identify useful contents.

b. If we ignore the title of the pages, we expect the object region to appear in contiguous area, both visually on the screen and in raw source of web pages. However, although the description for each object instance is visual contiguous, the raw source of such objects might not be contiguous.

c. If we compare similar pages from the same site using HTML elements as the unit of measure, we would find that most of the content differences appear in the object region, which is used to describe different object data with different attribute values. Thus by choosing a suitable set of content features, we should be able to identify irrelevant components as having similar structure and/or content, while object regions containing product information as different.

Based on these observations, we devise the following steps to extract object data (or product descriptions) and object model from complex web pages.

a. We employ spiders to download all related pages from a special site, or collect all the related pages from the service providers. We denote this set of web pages collected as P_{SET}.

b. We generate the DOM structures [9] of all the pages and use them as the logical structures to compare the similarities of web pages. For each node of the DOM structure, we simply use the text fragments as the content representation of that node. We use the tree-based kernel [7][12][23] to evaluate the distance between the web pages, as it could reflect the similarities in both the structure and content. We select any page p_j from P_{SET} and calculate the kernel $K(p_i, p_j)$. Because the pages similar to p_i are created by the same dynamic program or template, they should have similar structure and share many common tags among the nodes. Thus, it is not difficult to setup the threshold τ for similar page selection. The set of the pages similar to p_i is:

$$P_{SIM} = \{ p_j \in P_{SET} \mid K(p_i, p_j) > \tau \} \qquad (1)$$

c. From the DOM structures of the web pages, we expect the subtrees correspond to object regions to have the largest difference as they have distinct descriptions about different object data, while those subtrees correspond to irrelevant contents to be similar across the pages. To identify object regions, we compute the novelty value (which is a recursive definition of tree-based kernel, see next Section) for each subtree. The subtree that has the maximal novelty is the object region.

d. Next we detect object data within the object region. This is done by detecting the structure representation (see Fig. 3) within the object region. If there is more than one object involved, we partition them into different object data and output as different XML files as described in Section 5.

e. Given the set of object instances, we construct the object model as follows: (i) we construct some initial models by removing the repeated element among the object data; (ii) we integrate these initial object models; and (iii) we identify the data type of elements, detect the enumeration members and refine the object model.

f. Optionally, we seek domain experts to review and refine the object model learnt. The modified object model is used to verify object data found, design the database for object storage, and improve the performance of object extraction.

The above steps are summarized in Fig. 2. Here only step (f) needs human efforts to verify the object model. The following Sections describe the details of steps (c-e).

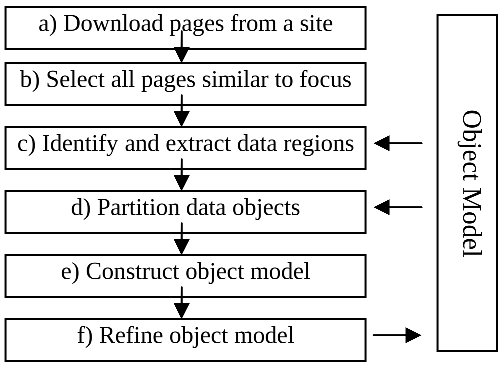

a) Download pages from a site
b) Select all pages similar to focus
c) Identify and extract data regions
d) Partition data objects
e) Construct object model
f) Refine object model
Object Model

Fig. 2 The main workflow for object model learning

4. DETECTING OBJECT REGION AMONG SIMILAR WEB PAGES

4.1 Features of object region

We first employ a DOM parser to generate the DOM parsed trees from the web pages, where each node in this tree is an HTML element. Object region is then a subtree that contains the description of the desired object data. For example, the object region could contain technical features about the notebooks, a table listing of staff, or stock information etc. The characteristic of object region is that it is typically the largest contiguous region in the web page having distinct tokens that are used to depict distinctive objects or product information. Thus we define the concept of repeatability for nodes and subtrees in the parse trees in order to identify object regions.

Repeatability is similar in concept to tree-based kernel. It defines the similarity between nodes or subtrees in terms of both contents and context. Here, similar node content means that both nodes share many content tokens and attributes of HTML elements; and similar node context is simplified as having similar parents. From the concept of repeatability, we define the concept of novelty of node or subtree in order to identify object regions.

4.2 Formalization

Definition 1: The content similarity of nodes n_1 and n_2 is defined as the weighted ratio of their shared tokens and attribute tags in HTML elements. Supposed that nodes n_1, n_2 have tokens t_1, t_2 and attributes a_1, a_2 respectively, where the number of tokens and attributes are denoted by |.|; and the number of shared attributes is $s(a_1, a_2)$. Thus the similarity of nodes n_1 and n_2 is:

$$Sim(n_1, n_2) = (\log(|t_1|) + 1) s(t_1, t_2) + w_1 \cdot s(a_1, a_2) \qquad (2)$$

Here, w_1 is the weight of the shared attributes and is set to 1 in our study. $s(a_1, a_2)$ represents the ratio of shared attributes or the proportion of common HTML tags in nodes n_1, n_2. $(\log(|t_1|) + 1)$ gives the weight of shared tokens $s(t_1, t_2)$ which has larger value when the number of similar tokens is higher. $s(t_1, t_2)$ is the similarity of tokens defined as:

$$s(t_1, t_2) = \begin{cases} 1 & if\ t_1 = t_2\ is\ idential\ in\ string-matching \\ .5 & if\ t_1, t_2\ belong\ to\ the\ same\ type \\ 0 & otherwise \end{cases} \qquad (3)$$

t_1 and t_2 belonging to the same type means that they are of same semantic type such as number, currency, e-mail and so on.

Definition 2: Supposed nodes n_1 and n_2 (not roots) are in parse trees T_1 and T_2 respectively, the repeatability of n_1 with respect to T_2 is

$$R(n_1, T_2) = \max_{\forall n \in nodes\ (T_2)} (sim(n_1, n) + w_2 \cdot sim(parent(n_1), parent(n))) \qquad (4)$$

where *parent*(.) denotes the parent node of n; and w_2 reflects the influence from the context of nodes n_1 and n_2, which is set to 0.5 in our system.

From Definition 2, two subtrees will have high repeatability if they are similar in both contents and context (or structure). In order to avoid cases where some object data sharing many similar descriptions that will increase the repeatability of the object region, we consider a window of N (i.e. N = 3~5) pages when evaluating the repeatability of the subtrees. We randomly select N pages from P_{SIM} and compute the average repeatability of their corresponding parse trees as the final evaluation measure. As repeatability of n_1 does not depend on the particular tree T_2, hence it can be simplified as $R(n_1)$.

We can normalize $R(n_1)$ by dividing it by $R(n_1, T_1)$ as:

$$\overline{R(n_1)} = \operatorname*{avg}_{T_i \text{ in Windows}}^{T_i \neq T_1} R(n_1, T_i) \Big/ R(n_1, T_1) \qquad (5)$$

Here $\overline{R(n_1)}$ is between 0 and 1. And then we could get,

Definition 3: The Novelty of node n_1 is

$$N(n_1) = 1 - \overline{R(n_1)} \qquad (6)$$

Definition 4: If *ST* is a subtree in parse tree *T*, the novelty of *ST* is

$$N(ST) = N(root(ST)) + w_3 \cdot \sum (N(ST_x)) \qquad (7)$$

where *root*(*ST*) denotes the root of ST; and ST_x is a subtree rooted at child node of *root*(*ST*). *N(ST)* is calculated recursively based on Formula (6). w_3 (set to 1 here) gives the contribution of novelty from all the subtrees. Thus *N(ST)* is the sum of the novelty of the nodes involved.

The subtree *ST* covering the object region should have the highest novelty value. Based on Definition (6), we expect the parent and higher level nodes of subtree *ST* to also have the largest novelty value among its siblings. Thus we expect the root of the smallest subtree containing the object region to exist along the path from the root of the overall DOM tree to a particular leaf whose nodes have the maximal novelty value among their siblings. Furthermore, as all other non-data-region subtrees should have low novelty values except for the subtree *ST*, we expect $N(n_i) > ½*N(parent(n_i))$ for the parent and higher level nodes of *ST*, and $N(n_c) <= ½*N(ST)$ for all child nodes n_c of *ST*. This leads to Definition 5 as:

Definition 5: Object region for DOM tree T_1 is the subtree *ST* having the largest novelty value, and that $N(n_i) > ½*N(parent(n_i))$ for the parent and higher level nodes of *ST*, and $N(n_c) <= ½*N(ST)$ for all child nodes n_c of *ST*.

4.3 Algorithm

Based on the above discussion, we design an algorithm to detect object region as follows.

```
GetDataRegion (Current_Parsing_Tree $T_0$, Parsing_Tree_Set
      {$T$}, window_size $N$)
{
1.   Randomly select $N$ trees $T_1, …, T_N$ from {$T$};
2.   For each (node $n_i$ in nodes($T_0$))
3.   {
4.        Calculate $R(n_i, T_k), k=1, …N$;
5.             $N(n_i) = 1 - \frac{1}{N-1} \sum_{k=1,\dots,N}^{k \neq i} R(n_i, T_k) \Big/ R(n_i, T_i)$;
6.        Recursively calculate Novelty for subtrees rooted at $n_i$;
7.   }
8.   while (navigating from the root of $T_0$ to a particular leave
     along the corresponding subtrees having the largest
     Novelty)
9.   {
          // parent ($n_i$) is the root of subtree containing the
          object region according to Definition 5
10.       if ($N(n_i) < N(parent(n_i))$ /2)
11.            return parent($n_i$);
12.  }
```

The computation cost for this algorithm is $N*|T|^2$, where *N* is the length of window, and |*T*| is the number of nodes in the parse tree.

5. PARTITIONING OBJECTS IN OBJECT REGIONS

5.1 Object data representation structure

There are five typical representation structures for objects as shown in Fig. 3. Of course, some pages may contain combinations of the above structures, such as a horizontal tabular structure (2b) embedded in a listing structure (2c) for a person's CV. Fortunately most object regions use only one type of representation. This is especially so for object regions in product web pages automatically generated by dynamic programs in most commercial sites.

	CPU	HD	Screen
Satellite 1000	PIII 800M	20G	14.1
Satellite 3000	PIII 1.5G	40G	14.1

(a) Horizontal tabular structure

	Satellite 1000	Satellite 3000
CPU	PIII 800M	PIII 1.5G
HD	20G	40G
Screen	14.1	14.1

(b) Vertical tabular structure

Toshiba 1000 - CPU PIII 800M - HD 20G - Memory 256M - Screen 14.1
Toshiba 3000 (optional) - CPU PIII 1.5G - HD 40G - Memory 512M - Screen 14.1

(c) (Nested) List

Toshiba 1000	CPU PIII 800M; HD 20G; Memory 256M; Screen 14.1

(d) Fragment structure

Toshiba 1000, 256 MB PC2100 DDR RAM up to 2048MB….

(e) Sentence

Fig. 3: Typical representation of extracted objects

If a set of objects are represented in different pages with different structures, it would be difficult to develop general wrappers to extract information from them. Thus the first task is to separate object data belonging to different products in order to simplify the task of extracting detailed attributes of such objects.

Each object instance is stored in an XML output file. An example of the XML output format is:

```
<Object>
    <Title>Satellite 1000</Title>
    <CPU> PIII 800M</CPU>
    <HD>40G</HD>
</Object>
```

Here Title is the object name.

5.2 Algorithm for object partitioning

If there is only one object involved in a object region, the transformation from object region to XML object output file is straight-forward. However, when multiple objects appear in the object region with one of the typical organization styles as shown in Fig. 3, we employ the divide and conquer strategy with heuristic knowledge to detect the number of object data in each object region as follows.

- Tabular structure. We first use the method reported in [26] to regulate the unified cell tagged by "Colspan=n" or "Rowspan=n". We compare the coordinates of cells in each column and each row. Since the same attributes about different objects will be aligned either vertically or horizontally in display, we could use the coordinate values to determine whether it is the horizontal or vertical tabular structure. If the average similarity among the nodes along the column is greater than that along the row, it means that it is the vertical representation structure. For vertical structure, we simply compare the first column with the other columns, and verify that the cells in the first column are about proper attribute names or general attribute values. The steps for constructing object data in horizontal tabular structure are similar.
- (Nested) Listing structure. In order to find out whether it contains only one object or a set of neighbouring objects, we need to investigate the repetition within the object region. If we can find similar subtrees that cover most of the object region, they will be used to generate different object data. Otherwise, we consider the whole object region as one data object and hence one XML output file will be generated.
- Fragment and Sentence structures. We assume that such region contains only one object. We parse the whole fragment as an entity in an XML output file.
- Compound structure. It is the combination of the above cases. It is usually maintained by human being. Fortunately for such intricate type, there is usually only one object involved in most cases. Our strategy is simply to parse the entire parse tree into an XML output file.

The following is the pseudo codes for partitioning object data from object region. Here t_1 is the threshold for selecting the candidates of attribute names; and t_2 is the threshold for differentiating the objects in listing structure.

```
PartitionObject(Current_Data_Region_Tree $T_0$,
    threshold $t_1$, $t_2$)
{
1.  Foreach (node $n_i$ in nodes($T_0$))
2.  { if ($R(n_i)$) > $t_1$)
3.     { Tag $n_i$ as attribute_name;}
4.  }
5.  if ($T_0$ is tabular structure)
6.  {
7.      Regulate the table;
8.      Detect the tabular representation direction -- vertical or
        horizontal;
9.      Combine the pair of corresponding cell and attribute name
        as an XML elements;
10.     Parse the first column or row as title;
11. }
12. else if ($T_0$ is listing) //one object in this case also
13. {
14.     For each (node $n_i$ in the second layer nodes of $T_0$)
15.     {
16.        Calculate the inner repeatability $R(n_i, T_0)$;
17.        if ($R(n_i, T_0) > t_2$)
18.            {Parse the subtree rooted at $n_i$ as XML output;}
19.        else //only one object
20.            {Parse $T_0$ as XML output; break;}
21.     }
22. }
23. else  //other structure
24. {   Parse $T_0$ as XML output; }
}
```

6. CONSTRUCTION OF OBJECT MODEL

In order to extract the abstract object model from the set of data object instances, we need to find the essential and common framework among all data object instances. It can be considered as a kind of the reverse engineering process to discover the ontology from XML documents. There are many tools, such as Protégé[1], xmlspy[2], Visual Stdio.NET[3], that can be used to generate DTD or schema for a single data object represented in XML. Here, we address the problem of deducing the object model based on the characteristics of the XML outputs of object data.

6.1 Identifying repeated elements in object data

An example of a complex XML data object output is shown in Fig. 4, which shows a number of repeated element (such as *Mouse*) or category (such as *Harddisk*). These repeated elements in the same category will share a common data schema, since they are the instances of the same element. Similarly, the repeated categories would reflect a common data schema also. However, the elements having the same element name but in the different categories might not share the same schema. For example, the element *Cache* in category *Processor*, and the element *Cache* in category *Harddisk* are two distinctive objects.

[1] http://protege.stanford.edu/

[2] http:// www.altova.com

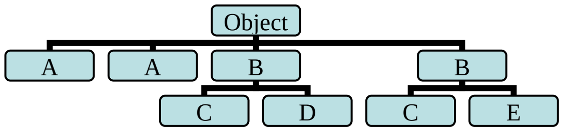
[3] http://msdn.microsoft.com/xml/

The data object shown in Fig. 4 can be represented as abstract model in Fig. 5(a). By identifying repeated elements and categories, we can derive initial object models as shown in Fig. 5(b). Here two elements with the same name exist in the same category will imply that they share the same schema. We can simply scan the simple elements in each category to check for repetitions, and combine them into one element tagged with “+” (where the occurrence is repeatable 1 to ∞ times) in the schema.

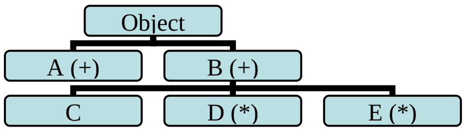
More work needs to be done to process repeated categories, whose children may not be completely overlap although their roots have the same name (see Fig. 4 for category *Harddisk*). We assume that the vocabulary in a site is explicit, thus those sibling categories sharing the same name should refer to same kind of entity, even though they might have different sub-elements. Hence we can union their child elements after we combine the repeated category. We will assign the tag “*” (meaning the occurrence repeats 0 to ∞ times) for the non-mandatory elements (see Fig. 5(b)).

```
<object name="computer">
    <Mouse>Inner mouse panel</Mouse>
    <Mouse>USB external mouse</Mouse>
    <HardDisk>
        <Type>IDE</Type>
        <Size>60G</Size>
    </HardDisk>
    <HardDisk>
        <Type>SCSI</Type>
        <Size>30G</Size>
        <Cache>1024k</Cache>
    </HardDisk>
</object>
```

Fig. 4: XML data object representation for *Computer*



(5a) A object instance (A, B, C… are element names)



(5b) The initial object model from (a)

Fig. 5: Deduce the initial data model from an object instance

6.2 Integrating the initial object models

Although most of the pages exist in our research are produced by dynamic programs or templates, the resulting object data may not have exactly the same structure. This is because the authors may delete the row when its corresponding optional element value is missing, and the number of repeated elements may be different. Thus we need to integrate these initial object models into a uniform one in order to eliminate the difference among data object instances and to enable all of them to be stored in the same database schema. We employ the following algorithm to produce the uniform framework for object data.

```
Integrate_Init_Ojbect_Model
    (Initial_Model_Set $M=\{M_1,\ldots,M_N\}$)
{
1.   $i = \arg\max_{i\in N} |M_i|$; //M_i has maximal nodes
2.   //update the tags of $M_i$'s elements
     For each node $n$ in $M_i$
3.   {
4.       if (any model in $M$-$\{M_i\}$ does not have the corresponding $n$)
5.           {Label $n$ “*”;}
6.       else if (the corresponding element of any model in $M$-$\{M_i\}$ is tagged “+”) //repeated element in $M$-$\{M_i\}$
7.           {Label n “+”; }
8.   }
9.   //add the elements which do not exist in $M_i$
     For each node $n$ in $M$-$\{M_i\}$
10.  {
11.      if ($M_i$ does not have a corresponding $n$)
12.      {
13.          Add $n$ to the corresponding position in $M_i$;
14.          Label $n$ “*”; //$n$ is optional
15.      }
16.  }
}
```

Here we assume that two nodes belong to the same element (or category) in two models if: (a) they have the same name; and (b) their parent node (category) has the same name. In the first step of this algorithm, selecting a data object instance with the maximum number of nodes helps to reduce the operations in adding the optional elements.

6.3 Refining the object model

We can extract more knowledge to refine the object model. One important issue is the identification of the type of elements. This is a crucial information for database design and object comparison. The other issue is in obtaining the enumeration list of elements. For example, the element “*Shooting modes*” in the camera domain would have enumerated values of: “*Auto, Manual, Long Shutter, Macro, Infinity, Stitch Assist, Movie*”, rather than a random set of strings. Most ontology learning approaches focus more on acquiring the definitions (or meanings) of object data. Here we focus on extending the element model (possible values and their aliases) in the concept space, as it would be of greater benefit to data understanding and reasoning.

The first problem is to identify the data type of the elements. Our system adopts the standard set of data types as described in W3C[4]. There are 36 kinds of recommended data type for use

[4] http://www.w3.org/TR/owl-guide/

with OWL, such as string, date, time, int, long et al. Here use the regular-expression and dictionary to detect the main data types: number (int, long, float, decimal et al), currency, data, time, language, Boolean and e-mail. We assign hierarchy to data types such that int is more specialized than decimal, and decimal is in turn more specialized than string, etc. The most general and default data type for element is the string. If the element in all data object instances contains only values of the more specialized data type, we would set its data type to the specialized one.

The second problem is to determine the list of enumerated values for the enumerated data types. Theoretically, we could use the frequency of all vocabularies for each element in the data object instances to enumerate its possible values. However, some elements may use arbitrary strings rather than based on a predefined set of enumerated values. To overcome this problem, we employ a pessimistic strategy to constrain the length of enumeration string to only one token, namely, the element cannot be enumerated if the length of its value exceeds one token. At the same time, we consider the token distribution, where most of the correct enumeration members should be repeated in many instances. A larger collection of data object instances would produce a more reliable statistics for this purpose.

The third problem is to detect element data format of the form: number follow by UOM (Unit of measure) such as "*40 G*" and "*800 MHz*" etc. Once we can correctly assign the element to this data format rather than just as string, we can perform computation on these elements according to some business logic. We again use regular expression to detect such data format with two attributes, quantity and UOM. Since we do not use any NLP parser to analyze the element data entity, we cannot deal with the complicated cases such as "*PIII 800 MHz*". This is the one direction of our future investigation.

7. EXPERIMENT AND DISCUSSION

Our experiments try to learn the data object model through detecting and constructing object data from complex web pages about products, such as the technical data for notebook and digital camera. This will facilitate answering of questions frequently asked in real applications that the users expect comprehensive answers involving tabulated results from web sites.

As this is a new area, it is hard to find a publicly available test corpus to test our technique. Current available corpora on faculty category, product category, and search snapshot are relatively well-structured with simple contents. Moreover, they assume fixed answer templates. In this research, we expect to extract detailed object level information (e.g. personal detail) with flexible template structure, rather than just category level information (faculty category) with fixed template structure. For these reasons, we need to build our own set of test corpus, comprising more abundant contents in product information. We therefore collect pages regarding notebooks, computers and digital cameras from different retailers and review sites as listed in Table 1. From these sites, we select 500 pages (with at least 10 pages from each site) that contain detailed descriptions of the target objects. About 98% of pages present product information in the form of tabular or list structures, while the rest present products in the form of text fragments or sentences. Some samples and results can be downloaded from http://www.comp.nus.edu/~yesr/webmining.

Table 1: Sites used for downloading product pages

http://www.pcworld.com/reviews/chart_test_report
http://sg.hardwarezone.com/priceguide/cat.php
http://www. amazon.com
http://www.gateway.com/home/products
http://www.csd.toshiba.com/cgi-bin/tais/pc/pc_home.jsp?comm=ST
http://list.auctions.shopping.yahoo.com/23336-category.html?alocale=0us
http://www.nextag.com/Notebooks~300359z0zBwzmainz5-htm
http://www.kodak.com
http://www.canon.com
http://www.sony.com
http://www.nikon.com/
http://www.olympus-global.com/en/global/
http://www.jvc.com/

For each web page, we built the DOM parse trees using the HTML DOM [9]. The kernel method for retrieving similar pages from the same site can achieve quite high performance. This is because there are high similarities between the pages produced by the same program, and they are very dissimilar to other pages that are hand-coded or derived from different programs. Because of the limitation of paper length, here we focus only on evaluating three most critical steps in our framework – the quality of the extracted object region, the accuracy of partitioning the object data within the object region, and the performance of constructing the object model.

7.1 Test on object region extraction

Since the object region is a subtree of the parse tree in the corresponding page, we can evaluate the quality of the detected object region in two aspects: (a) whether it covers all nodes depicting the object data; and (b) whether it contains irrelevant components. The intuitive approach is to compute the overlapping degree between the test object region and the ideal object region using recall and precision. However, because the cost of missing the nodes about the objects is much higher than that of covering irrelevant nodes, hence recall is more important than precision. Fortunately, missing of data object nodes nearly never happen in our testing, although there is an average of 1~6% of nodes that are not relevant to the object data directly. We found that this is caused by the definition of object region. For example, if there are three nodes a, b, and c in the second layer of the object region, two subtrees A, B rooted at a and b are about two object data, but a small subtree C rooted at c

XML Schema Part 1: Structures Second Edition http://www.w3.org/TR/2004/PER-xmlschema-1-20040318/

contains irrelevant content (such as a link to the promotor's page). We had to include subtree C in the proposed object region if we want the subtree to cover A and B. This problem only happens in pages formatted using listing structure. These pages are likely to be manually constructed. They thus do not have high quality structure since the nodes about the object data are not encapsulated, and are mixed with irrelevant nodes. But these irrelevant nodes are likely to be ignored in the following object partitioning step.

7.2 Test on data object partitioning

In the step of partitioning objects, we divide the object regions into XML output; each output corresponds to a data object. Table 2 gives the statistics of the test corpus in terms of the distribution of data representations and the number of unique object data or output files. The performance of partitioning objects for each unique data object is tabulated in Table 3, which shows an average F_1 of 94 %.

Our error analysis reveals the following sources of errors. One typical error occurs when analyzing the structure of raw table (vertical and horizontal tabular) entries involving unknown attributes, which are being used to construct spurious object data. Another source of error is that the system mistakenly partition one object into many objects, where the content features of the grouped attributes are very similar to each other. The other major source of error is the nested list structure, as the quirky similarity between the subtrees might mislead segmentations.

As compared to the size of original source pages, the size of data object's output decreased tremendously, indicating a great reduction in irrelevant elements. For example, one digital camera technical page in our testing corpus reaches 556KB with more than 5,000 parse nodes, while the data object file is only 8 KB with about 100 XML elements.

In general, we found that we are able to extract all object data from the well-structured pages with no missing cases but with some incorrect detection. This conclusion is similar to that reported in [18] on very simple cases. Also, our overall performance of constructing object data is close to that achievable in object mining and extraction as reported in Omini [5], but we could process more complicated structures such as nested listing and fragments.

Table 2 Object data distribution

Type	Representation	# of pages	# of object data
1	Vertical Tabular	400	637
2	Horizontal Tabular	30	74
3	Listing	70	97
4	Text Fragment	6	8
5	Sentence	4	4
6	Combined	0	0
TOTAL:		510	820

Table 3: Performance of partitioning object data

Type	Correct	Incor-rect	Miss-ing	Precision (%)	Recall (%)	F1 (%)
1	621	34	16	94.8	97.5	96.1
2	74	5	0	93.7	100	96.7
3	76	14	21	84.4	78.4	81.3
4	7	3	1	70	87.5	77.8
5	4	4	0	50	100	66.7
Avg.	782	60	38	92.9	95.4	94.1

7.3 Test on data object model construction

For this test, we focus on constructing the data object models for digital cameras based on web pages as listed in Canon and Kodak web sites. We employed the algorithms discussed in Section 6 to construct the data object models based on the relevant data object instances extracted. Table 4 and Table 5 respectively list the models extracted from 17 data object instances about Canon digital cameras and 14 data object instances about Kodak digital cameras. Fig. 6 shows the diagram of object model about Kodak digital camera. For comparison purposes, we include in Fig. 7 the manually constructed ontology diagram of (digital) camera[5].

From Tables 4-5, and Figures 6-7, we can draw the following observations.

a) There are nearly 100 elements defined in the object model for Canon camera, and 46 elements for Kodak camera. However, the ontology created by human experts has only about 30 elements (including attributes). The overlapping of elements among these models is very low, indicating that the existing ontology might not match the data that we face in real web pages. It is difficult to apply the available object model and ontology to the particular data. Thus we need to construct the updated data object model for these data, so that it can be used to help in extracting, storing and utilizing data object models.

b) Both of our constructed object models use 3-layer hierarchical structure, but the element organization is quite different, where they have 23 and 4 categories respectively. Both models organize their elements according to different principles and could present their content using different frameworks. The semantic among the categories and their elements are related, consistent and reasonable, and we cannot say which one is better.

c) The name of elements used in the two object models might be different even though they refer to same descriptions. Another variation is the difference in granularity among elements, where one element in a model might be split into multiple sub-elements in another model. Thus it is not easy to compare the element using different model even though they are about the same type of products. How to construct the universal ontology based on the available object models is one of our next research topics.

d) The number of repeated elements in Kodak model is nearly one quarter, but we cannot find the case of repeated category.

[5] http://protege.stanford.edu/plugins/owl/owl-library/camera.owl

This generalization could benefit the flexibility in data storage and XML query.

e) For this test, we do not correct the error in data object instances before performing model learning. As a result, there are 3 spurious elements exist in the above two models.

f) It is also observed that there are few elements with data type beyond the type *string* in our test corpus. The existing data models contain data of type decimal, int, Boolean, and country. We also extract the proper attributes (amount and UOM) for elements such as "*16mm*". However, we cannot process the more complicated cases such as "*Approx. 165g*" and "*85.0 x 56.0 x 23.9 mm*", since there is not NLP parser component in our current system.

Table 4 Object model for Canon digital camera

Category	Element
IMAGE-SENSOR	Type, Effective-Pixels, Total-Pixels, Aspect-Ratio, Color-Filter-Type
IMAGE-PROCESSOR	Type
LENS	Focal-Length, Zoom, Maximum-f-number, Construction
FOCUSING	Type, AF-System-Points, AF-Modes, AF-Lock, AF-Assist-Beam, Closest-Focusing-Distance
EXPOSURE-CONTROL	Metering-Modes, AE-Lock, ISO-Speed-Equivalent
SHUTTER	Type, Speed*
WHITE-BALANCE	Type, Settings
COLOUR-MATRIX	Type
VIEWFINDER	Viewfinder, Eyepoint
LCD-MONITOR	Monitor, Coverage, Brightness
FLASH	Modes, Slow-Sync-Speed, Red-eye-Reduction, Flash-Exposure-Lock, Built-in-Flash-Range
SHOOTING	Modes, Photo-Effects, Drive-Modes, Continuous-Shooting
RECORDING-PIXELS - COMPRESSION	Image-Size, Compression, Movies, Movie-Length, Movie-Length
FILE-FORMAT	Still-Image-Format, Movies, Sound-Files
DIRECT-PRINT	Canon-Printers, PictBridge
OTHER-FEATURES	My-Camera, Sound-Memo, Intelligent-Orientation-Sensor, Histogram, Playback-Zoom, Image-Erase-Protection, Image-Erase, Self-Timer, Menu-Categories*, Menu-Languages, Firmware-Update
INTERFACE	Computer, Other
MEMORY-CARD	Type
OS	PC, Macintosh
SOFTWARE	Browsing-Printing, Other, Drivers
POWER-SOURCE	Batteries, Battery-Life*, AC-Power-Supply
ACCESSORIES	Case, Power-Supply-Battery-Chargers
PHYSICAL-SPEC.	Body-Materials, Operating-Environment, Dimensions, Weight-body-only, Continuous-Shooting, Movie-Length

Table 5 Object model for Kodak digital camera

Category	Element
Standard-Features	CCD-resolution, image-resolution, image-quality*, compression, zoom*, focus-auto-focus, focus-distance*, display*, aperture*, shutter-speed, ISO-equivalent*, white-balance, flash-mode, flash-range*, self-timer
Performance-Features	scene-other-modes, color-mode, macro-close-up-mode, burst-mode, light-metering-method, exposure-compensation, exposure-control, movie-mode, movie-image-resolution, movie-length
Smart-and-Simple-Features	capture-mode, auto-orientation, delete, review, share, video-out, software, interface
Additional-Features	Storage*, power-options, image-file-format, interface, lens-protection, tripod-mount, weight, dimensions, warranty

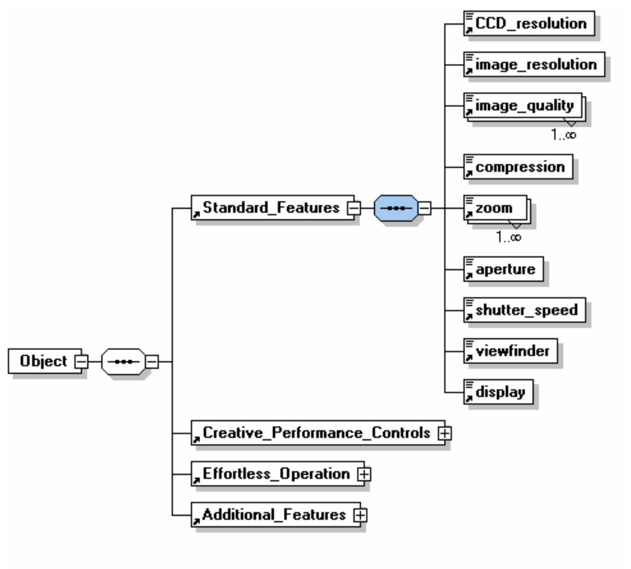


Fig. 6: Data object model for Kodak digital camera

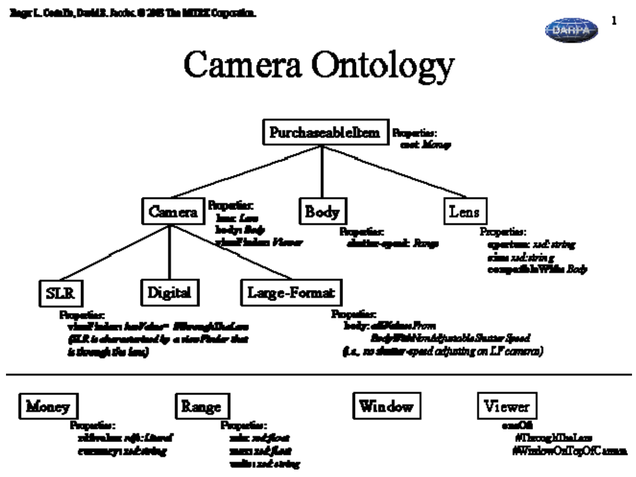


Fig. 7: Ontology created by domain experts (from http://www.xfront.com/camera/sld001.htm)

8. CONCLUSION

Ontology learning is the key technology to realizing the ideal of semantic web and to support business intelligence. This paper focused on tackling the problem of learning the domain ontology from complex web pages, where the presentation structure is varied and the useful data is surrounded by many irrelevant components. Our tests demonstrated that our system is effective and efficient.

The main contributions of our research are four-fold. First, we defined the similarity measure between the nodes and constructed the novelty measure to evaluate the repeatability of nodes or subtrees appearing in the selected page set. These measures can be used to effectively reflect the essential features in the irrelevant components, object regions, and the description framework of object data.

Second, we proposed an automated approach to identify object region, which is the smallest subtree of the parse tree covering the descriptions of all desired object data. Further processing of object region is therefore free from interference from irrelevant components.

Third, we investigated the algorithm for partitioning the object data in object region. This algorithm could recompose the non-contiguous description about different object data and produce the independent self-explainable XML output files for the objects.

Fourth, we explored the method for inducing the abstract object model from the data object instances. The object model is the generalization of the original data and is the key to understanding and using the object data.

To improve the performance of our system in processing complex pages, we need to further improve and fine-tune techniques for enriching the detected data object model, and extracting distinct object data from object regions. We also need to build a larger test corpus and methodologies to test our new framework. Finally, we would like to employ NLP technologies to learn deep semantic information from the pages and build more comprehensive ontologies.

9. REFERENCES

[1] Assadi, H. Construction of a regional ontology from text and its use within a documentary system. FOIS-98, 1998.

[2] Bar-Yossef, Z. and Rajagopalan, S. Template Detection via Data Mining and its Applications, WWW 2002, 2002.

[3] Bunge, M.A. Treatise on Basic Philosophy: Vol. 3: Ontology I: The Furniture of the World. Reidel, Boston, 1977.

[4] Buttler, D., Liu, L., and Pu, C. A fully automated extraction system for the World Wide Web. IEEE ICDCS-21, 2001.

[5] Buttler, D., Liu, L., et al. OminiSearch: A method for searching dynamic content on the Web, ACM SIGMOD 2001.

[6] Chang, C-H. and Lui, S-L. IEPAD: Information extraction based on pattern discovery, WWW-10, 2001.

[7] Collins M. and Duffy, N. Convolution Kernels for Natural Language, Advances in Neural Info Proc. Sys., vol(14), 2002.

[8] Doan, A., Domingos, P. and Levy, A. Learning Source Descriptions for Data Integration. WebDB-2000, pp 81-86, 2000.

[9] DOM, http://www.w3.org/TR/DOM-Level-2-HTML/ 2003.

[10] David W. Embley, Douglas M. Campbell, Randy D. Smith, Stephen W. Liddle. Ontology-Based Extraction and Structuring of Information from Data-Rich Unstructured Documents, CIKM, pp52-59, 1998.

[11] Esposito, F., Ferilli, S., Fanizzi, N. and Semeraro, G. Learning from Parsed Sentences with INTHELEX. Proc. of Learning Language in Logic Workshop, Lisbon, 2000.

[12] Gärtner, T. a Survey of Kernels for Structured Data. Newsletter of ACM SIGKDD. 5(1), Jul 2003.

[13] Gupta, S. Kaiser, G. et al. Adapting content to mobile devices: DOM-based content extraction of HTML documents, WWW-12, pp207-214, 2003

[14] Hahn, U. and Schnattinger, K. Towards text knowledge engineering. AAAI-98, pp 524-531, 1998.

[15] Holger M. Kienle and Hausi A. Müller. A tool-supported method to extract data and schema from web sites, 5th Int'l Workshop on Web Site Evolution, 2003.

[16] Lerman, K., Knoblock, C. & Minton, S. Automatic data extraction from lists and tables in Web sources. IJCAI-01, Workshop on Adaptive Text Extraction and Mining, 2001.

[17] Lin, Shian-Hua and Ho, Jan-Ming. Discovering Informative Content Blocks from Web Documents, KDD-02, 2002.

[18] Liu, B., Grossman, R. and Zhai, Y. Mining Data Records in Web Pages. KDD-2003.

[19] Kietz, J.-U. and Morik, K. A polynomial approach to the constructive induction of structural knowledge. Machine Learning, 14(2), pp193-217, 1994.

[20] Kushmerick, N. Wrapper Induction: Efficiency and Expressiveness, AI vol(118) pp15-68, 2000.

[21] Maedche, A. and Staab, S. Semi-automatic Engineering of Ontologies from Text. SEKE-2000, Chicago, 2000.

[22] Maedche, A. and Staab, S. Ontology Learning for the Semantic Web. IEEE Intelligent Systems, 16(2):72-79, 2001.

[23] Müller, K.R., Mika, S., et al, An introduction to kernel-based learning algorithms. IEEE Neural Networks, 12(2):181-201, 2001.

[24] Omelayenko B., Learning of Ontologies for the Web: the Analysis of Existent Approaches, In: Proceedings of the International Workshop on Web Dynamics held in conj. with the 8th International Conference on Database Theory (ICDT'01), London, UK, January 3, 2001.

[25] Schlobach, S. Assertional mining in description logics. In Proc of Int'l Workshop on Description Logics (DL-2000), pp 237-246. 2000.

[26] Shimada, K., Fukumoto, A. and Endo, T. Information Extraction from Personal Computer Specifications on the Web Using a User's Request, IEICE on Information and Systems, pp1386-1395, 2003.

[27] Studer, R., Benjamins, R. and Fensel1 D., Knowledge Engineering: Principles and Methods. Data and Knowledge Engineering, 25(102):161-197, 1998.

[28] Uschold, M. Knowledge level modeling: concepts and terminology. The knowledge Engineering Review, 13(1):5-29, 1998.

[29] Wand, Y. A proposal for a formal model of objects. In W. Kim and F.H. Lochovsky, editors, Object-Oriented Concepts, Databases, and Applications, pp 537-559. ACM Press, New York, 1989.

[30] Webb, G., Wells, J., Zheng, Z. An Experimental Evaluation of Integrating Machine Learning with Knowledge Acquisition. Machine Learning, 31(1): 5-23, 1999.

[31] Yi, L. and Liu. B. Eliminating Noisy Information in Web Pages for Data Mining, KDD-03, 2003.