

# Query Ontologies for Autonomous Online Resource Integration Systems

Maksym Petrenko, Hasan Jamil

**Abstract** — Tasks of autonomous online data integration to be adopted by average user require on-the-fly processing and rapid response. Precisely, user should be able to formulate queries over the heterogeneous autonomous online resources (usually - WEB pages) and get results in reasonable time. To achieve this we need to design light-weight techniques and tools that will be able to utilize effectively information provided by the user both implicitly (through queries) and explicitly (through user-defined or user-chosen ontologies). While the existent integration systems make extensive use of the former, the knowledge contained in the user query remains to the large degree unused and is utilized only on the last stages of the integration process. In this paper we investigate the approach of Query Ontology which utilizes both to minimize overhead caused by processing relatively big amounts of data, contained in each WEB page and irrelevant to user query, by making use of the knowledge, contained in the user query, on the earlier stages of integration process. We show how to reduce the computation complexity of ontology-based integration systems and propose the framework.

**Index Terms** — Query ontology, autonomous integration, information systems, schema matching, data extraction.

## I. INTRODUCTION

TODAY'S Internet provides repositories of information that can of interest for user in particular cases. While being extensive, information in the WEB repositories is usually stored in non-uniform, heterogeneous way. The problem of data heterogeneity is easily solvable by human; however, humans are just unable to process the huge amounts of data, stored in the Internet. Thus, automated data integration, as it was discussed in [1] for example, is severely needed to be able to use efficiently information stored in the Internet.

Present solutions to this problem (for example, the one proposed by D. Embly et al. in [2]) are usually based on the knowledge models – Ontologies ([3]) - and involve the next steps: extracting of data from web-pages using ontology dictionary; structuring data and putting it into the database using ontology structure; querying populated tables using specified query projected into the database scheme again through ontology. While providing sufficient degree of flexibility and automation, this approach produces a lot of

overhead in terms of both irrelevant data, “traveled” through the mentioned stages, and ontology manipulation operations, involved on each step of integration process. Even though the former can be neglected if the small web-pages are considered and processed, the latter still can be highly expensive in terms of computation complexity since precise ontologies require huge and complex graph data structures as well as operations of corresponding magnitude.

In this work, we propose and discuss the Query Ontology model. In a nutshell, this approach utilizes the tiny subontology computed from the full user ontology by truncating it in accordance to the user-provided query and the characteristics of ontology-based algorithms, used in the underlying integration processes. This helps us to push query answering task down through the described above data integration workflow and thus minimize required computations and data flow on each stage of the integration workflow.

Later in the paper, we discuss practical application examples of proposed query ontology concept and show how our approach can reduce the task complexity for different stages of automatic ontology-based WEB data integration systems thus improving overall response time and throughput of these systems. Still, we warn that, to be able to make profit of query ontology, ontology-aware components of the integration information system should not have the recursive properties; therefore, potential targets for the described approach are the tools that perform the operations over the ontology's terms in their relatively small local context rather than in context of the whole ontology.

As it was mentioned before, we make an overview of the query ontology concept in the context of autonomous integration systems and propose to utilize this approach in the large WEB-service providers' systems rather than on the users' desktop systems where the response time is not so critical. We also believe that used here analysis framework can be employed in other information systems. For example, it can be used to decide how to enrich with ontology terms user's search query in information seeking systems.

The rest of the paper is organized in the following way: section 2 describes the model of proposed query ontology approach; in the section 3 we discuss the use framework of proposed model; section 4 makes an overview of the two main components of autonomous ontology-based integration systems, specifically schema matcher and data extractor, and

---

Maksym Petrenko is with the Computer Science Department, Wayne State University, Detroit, MI 48202 USA (max@wayne.edu).

---

Hasan Jamil is with the Computer Science Department, Wayne State University, Detroit, MI 48202 USA (jamil@cs.wayne.edu).

discuss them in the context of query ontology on example of four available tools – PickUp[4], tool developed by Data Extraction Group of Brigham Young University [2], Cupid[5] and OntoBuilder[6]; section 5 makes overview of the related work; section 6 concludes and defines the area of future work.

## II. MODEL

### A. Ontology-based integration systems

Tasks of automatic online data integration are of great interest and development today. Due to the wide heterogeneity of the data, published on the WEB, the big subclass of WEB data integration systems is able to utilize the known model of the domain of interest called *user ontology* and *WEB-pages domain model* (derived from the analyzed WEB-pages) thus extracting the data available in the WEB-pages into the database, structured accordingly to the user specifications. Sometimes term *global ontology* is used instead of user ontology; even though global ontology is more correct term if the data integration is performed by some information service system, for this paper’s context there is no substantial difference between these term and in the rest of the paper we will use these terms interchangeably.

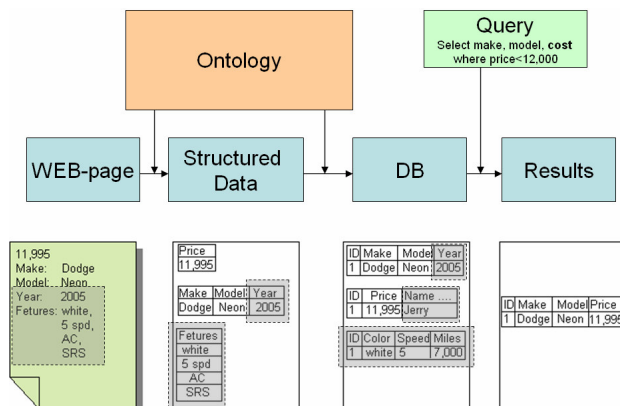


Figure 1. Autonomous WEB data integration process.

The typical integration process of ontology-based systems for semi-structured WEB data is depicted in Figure 1 and usually involves the next steps:

1. Extraction of tables or record-value pairs from WEB-page using global ontology dictionary.
2. Structuring of extracted information using global ontology relations.
3. Database population.
4. Querying of populated database with user query.

This process allows user to bring desired data from Internet to the specified databases and then work with it locally; moreover, thankfully to the user ontology used on the second step, extracted data will be presented to the user in the form that will reflect his knowledge model rather than in WEB sites domain’s model (please refer to [12] for explanations of global-as-view information system model).

While being suitable for some tasks, where all the available on-line data are required, described process literally “drugs” a lot of unneeded data through the whole information system if only the small specific part of data is desired. In the example from Figure 1, user queries for the maker, model and price of all the available cars that are priced fewer than 12000. Obviously, typical car advertisements contain far more auxiliary information (marked on the picture with gray dashed rectangles) like year of make, color, type of brake system, etc. Though user is not interested in this additional information, it still will be processed and carried through the whole system up to the population into database; finally, the unwanted data will be filtered out and dropped on the fourth step of the described process.

It is clear that in face of massive amounts of WEB-pages and data, stored in these pages, the computation operations required to process the WEB data will cost too much of operational time and space. Moreover, extensive and detailed ontologies are represented by the huge graph data containers and make the computations even more expensive.

### B. Query Ontology

To fight the problem of processing irrelevant data in the automatic online data integration systems, we propose the model, which pushes the task of user query satisfaction down through the integration system architecture. More specifically, for the data extraction and structuring task we are using *query ontology* which is the light-weight ontology, calculated from the available user ontology by matching it with user query terms and constrains; also, while constructing the query ontology, we consider the algorithms of the subroutines, which comprise the integration system workflow.

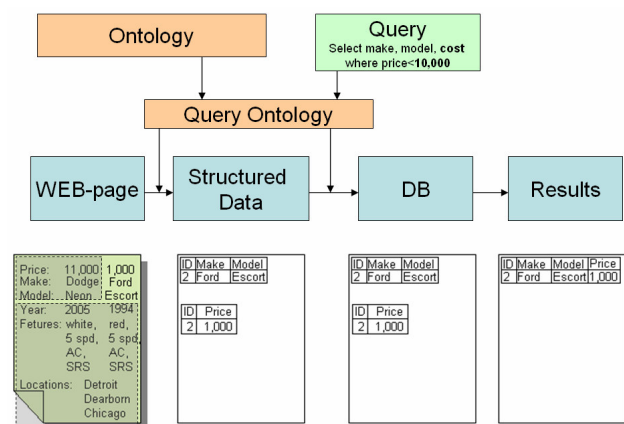


Figure 2. WEB data integration using query ontology.

Proposed data integration process model is pictured in Figure 2 and involves the next major steps:

1. Construction of query ontology from global ontology and user query.
2. Use obtained in the first step query ontology to tune-up subroutines that comprise the integration system service.

3. Application of tuned extractor to extract semi-structured data from WEB-pages.
4. Structuring of extracted data using query ontology relations and application of user query constrains.
5. Database population.

Here, the first step involves building query ontology which serves as the basis for the rest of the model's process components. Generally, query ontology is the ontology, built from the global ontology using the terms and constrains available in the user query (for this time, we assume that user query is expressed in terms of available global ontology). Further, query ontology includes not only the user-defined terms, but also all the terms needed to preserve the ontology structure within the scope of the query as well as the certain terms, needed for the correct work of the subroutines, that comprise the integration system service (this is discussed in details later in this paper).

On the second step, the integration system can tune-up its ontology-aware routines accordingly to the constructed query ontology. In the Examples section we show how we can tune-up Pick-Up extractor using query ontology. Briefly, we can, for example, tell WEB-page data extractor to extract only the terms that appear in the query ontology, or, additionally, we can set up the starting point of the data extraction tool within the WEB-page accordingly to the top term of the query ontology. All these efforts should make the system be able on the third step to extract efficiently only the data, relevant to the user query.

On the fourth step, system puts the extracted data into the form of the relations, derived from the query ontology structure by matching extracted data terms and structure with those in the ontology. Also, user query constrains can be applied on this step if they were not applied on the previous step (some data extractors process WEB-page data in bulk and thus can not be tuned to accommodate fine-grained query constrains).

Finally, system will populate the specified database with the extracted and structured data. At this point, populated tables should contain only the data terms (columns), specified by user in his query. As it described above, user constrains can also be applied on different steps of proposed model. However, for some information systems this can be hard or even impossible to implement. In this case, constrains can be put over the populated database as it is done in regular autonomous integration systems; still, those constrains will be processed only over the terms user is interested in.

### III. FRAMEWORK

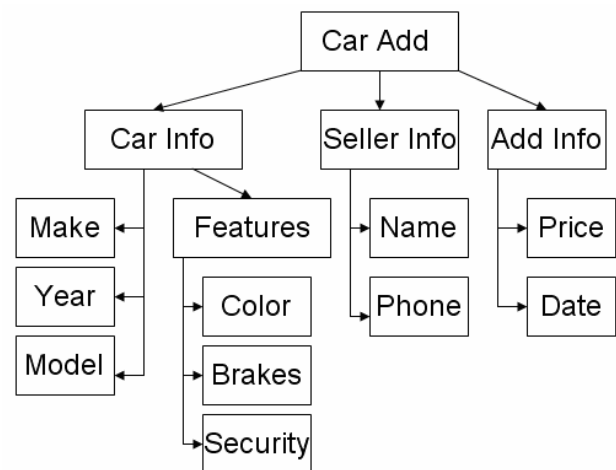
Since the query ontology didn't introduce major modifications into the architecture of the typical automatic online recourse integration system, it is possible to reuse existing systems with the slight modifications of the implementation. Here we give the general framework of this modification process.

From the Figure 2 one can see that the integration process, updated with the query ontology concept, to the large degree remained the same; therefore, general structure of existing integration system will remain almost unchanged. However, here are the two major challenges, which can lead to changes in the system's components. In our model, system uses query ontology – substantially smaller version of the initial user ontology. This means that the system will operate with the incomplete knowledge (if compared to the initial ontology), which, in turn, can affect the ontology-aware components (e.g. schema matcher, data extractor etc.) of the system. So, the main challenges in this context will be the next:

1. How to construct the lightest query ontology in the way, so it will be possible to use successfully it in the system's ontology-aware components.
2. If there is the need or possibility, how to modify the system's ontology aware components so they can work correctly and effectively with constructed query ontology.

Please note, that these two problems are mutually dependent, so the system architect can be required have to find the balance between query ontology size and the quality of service of system's ontology-aware components.

Even though there is all-purpose solution to these problems and instead the set of needed terms should be inferred from the underlying algorithms of the used subroutine, we give here common heuristics that should be considered when compiling query ontology.



**Figure 3. Sample car advertisements ontology.**

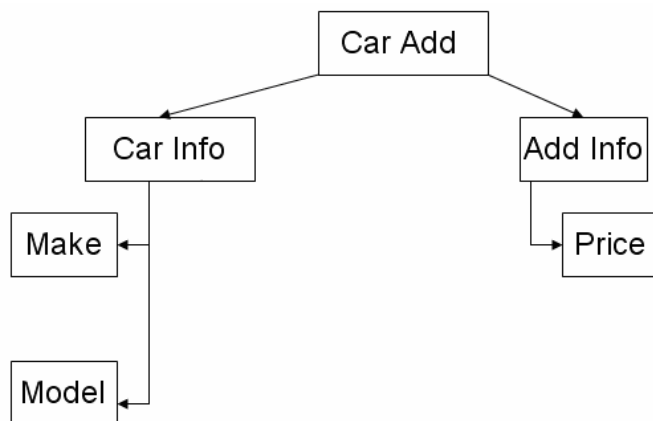
Before we start, recall that the ontology is the entity that is comprised of terms (things), their attributers (values) and certain relationships defined among the terms [7, 8]. Note, that this defines conceptual model rather than knowledge representation, used in AI research; therefore, it's possible for us to use partial rather than complete domain model representation. Figure 3 shows sample domain model that we built for the car classifieds from the Figure 2 example. Please note also, that even though ontologies can be represented by

graphs, typically they can be modeled by the correspondent trees as it done in [5] for example. Also, constrains specified in the user query can also be encoded in the tree structures as it was shown in the aforementioned work. Thus, in this work we will view the ontologies as the tree-like structure.

### A. Heuristics

Our first heuristic is that the one should obviously keep in the query ontology all the terms (together with their attributes) that user query for - without them use of the ontology will lose any sense.

Next, if we keep in query ontology only the elements of interest without any structural information, we will not be able to present user with the extracted integrated information in the structural way (e.g. in form of relations); instead, user will be able to see only the set of separate atomic elements, extracted from the pages. Therefore, one should keep in query ontology the structural dependencies among terms of interest. Since elements of interest can be kept in the different parts of the global ontology; therefore, relations among components of interest can't be preserved by only keeping the ontology tree edges that directly connects terms of interest and represents the structural dependencies among these components. Here, to preserve the structural integrity of the query ontology, we propose to keep all the terms along the tree paths starting from elements of interest up to their least common ancestor. The sample query ontology tree built for the example from Figure 3 using first two heuristics is shown in the Figure 4.



**Figure 4. Sample Query Ontology built using first two heuristics.**

It is also possible to further reduce the proposed tree structure by replacing all the intermediate terms user is not interested in by either:

- The edge from the leaf term of interest to another ancestor non-leaf term of interest if any.
- The edge from the leaf term of interest to the root of the query ontology tree otherwise.

However, we prefer to keep the all the ancestors of the elements of interest, since they can be needed in our next

heuristic.

Third heuristic involves the analysis of the particular ontology-aware subroutines used in analyzed information system. Usually, ontology-based tools operate with the ontology terms based on their linguistic form, their structural context or both. While linguistic attributes of the terms in the query ontology compared to the initial global ontology remain untouched, the structure changes substantially. Therefore, one should carefully analyze the nature and extent of the structural context, used to operate with terms. Examples of nature of structural context can be the next: parents and children of the term, leaf set of term, siblings of term, predecessors and successors of element etc. Respectively, if subroutine needs this context, one should preserve in query ontology the context of terms of interest. For example, if routine needs parent context, all the parents of the terms of interest should be saved in the query ontology.

Another important property of structure-aware subroutines is the extent of the needed context. For example, many tools require only the local context of the term – e.g. direct parent, direct children etc. In this case heuristic is simple – it is completely enough to preserve in query ontology only the local context of elements of interest. As the more complex example of local context we can name the leaf set of the considered term and subtree, rooted at element. In this case local context, saved in query ontology, can be bigger then in previous example depending on the width of leaf set and depth of subtree, rooted at the element of interest. Still, the query ontology tree in this case can be substantially smaller then the original user ontology tree.

Further, in certain cases only the part of the local context is need – i.e. the elements of local context, which total number doesn't exceed certain threshold (for example, one third of direct children). In this case it's possible to keep in the query ontology only the part of the local context which satisfies threshold value. Plus, it is possible to setup the threshold value to keep the desired balance between query ontology size and subroutine output quality, controlled by the threshold value. On the downside we can name increased complexity of logic, needed to evaluate the suitable thresholds' values.

The most complex case of structural context appears in case if subroutine algorithm has the recursive nature. In such algorithms each tree elements' characteristics can make impact on all the other elements' characteristics. Thus, removal of any single term will affect all the properties of all the other terms in the tree. For example, for well-known heap sort algorithm, the absolute positions of the elements obtained after sorting some tree can be totally different then those obtained after the sorting of the same tree having one element removed (though the elements will still be in the correctly sorted relative order). Typically, in such an algorithms local context weights more then global context, where the weight is controlled by the system of threshold values. In this case it is possible to find the such the combination of threshold values and the size of saved context, for which the algorithm will perform relatively

precisely (algorithm precision is fuzzy anyway since controlled by threshold). However, such the algorithms are hard to analyze and predict the behavior, so one should be ready to face these challenges if choose to use query ontology.

Our final heuristic suggest that with help of query ontology we can try to further automate both the ontology-aware and ontology-unaware tools that requires user input to specify some values, that can be implicitly derived from information, that contains in user query. For example, WEB-page data extraction tool can use terms specified in query (and thus saved in query ontology) as the pivot to identify correctly the structure of interest in the WEB-page. Also, if nested tables are considered, depth of query ontology can be used as the depth of table nesting. Unfortunately, such the properties are unique for each subroutine and thus are impossible to generalize here.

Overall, we argue that careful analysis of the integration system's subroutines in most case allows building the query ontology, which is substantially smaller then the original one. In the next section we list couple available data integration tools and explain our heuristics on example of these tools.

#### IV. EXAMPLES

Generally, any set of WEB data integration tools can make use of query ontology. In this section we show how to apply the proposed analysis framework on the two main components of integration systems - schema matching and WEB data extraction tools.

It's clear that schema matching process is important for the WEB data extraction process since there is the need to match different structures like user query, query ontology, information extracted from the WEB-pages, etc.

As it was mentioned before, global ontology is the entity that carries the knowledge about the data structure; therefore, on most steps of the autonomous extraction process system matches various data with the global ontology schema. Since the problem of schema matching is the problem of graph isomorphism (which is known to be NP-complete), there is no efficient solution to this problem; furthermore, differences in WEB schemas (i.e. different models for the same domain) introduce additional difficulty in this process. Thus, most of the schema matching algorithms propose some heuristic to produce solution approximation. It's clear that under this circumstances the smaller ontology is - the faster matching process can be performed. Respectively, to make use of query ontology one should consider our structural heuristics in the first priority.

WEB data extraction, in turn, is of the second (if not of the first) importance subroutine of automatic WEB data integration systems. Usually, ontology-based process of data extraction from WEB-pages consists of the two main tasks in respect to the domain vocabulary and domain ontology: terms extraction and terms structuring. While in case of unstructured WEB-page data extractors are tailored to the basic tokenizing, in case of structured and, more often, semi-structured data

extractors obtain the data from the pages together with certain structural information (e.g. extracted tables, lists, trees etc.).

Extractors of first case can be easily optimized to look for and process only the concepts that appear in query ontology. In turn, structure-aware extractors require the same careful analysis of the underlying extraction algorithm as it was suggested for the schema matchers. For example, if target structure is the table, the system can extract only the columns that appear in query ontology. Next, in case of tree-targeted structure, system can try to locate in the page (using regular IR techniques) root element of the query ontology and then start computation-intensive structure analysis from that located point.

While brief overview of data extraction tools appears in [9], the available schema matching approaches are summarized in [10]. For the illustration purposes, in this chapter we discuss the proposed framework use on the example of two available schema matching tools, namely *OntoBuilder* and *Cupid*, and two data extraction tools, *Pick-Up* and *BYU* tool. *Cupid* and *BYU* tools were chosen as one of the best representatives for their respective classes, while *OntoBuilder* and *Pick-Up* are more recent examples which don't appear in aforementioned surveys and claim to outperform *Cupid* and *BYU*. Solutions, discussed here, intended to be simple and can be used as the starting point for the particular tool set analysis. We also chose not to include here the analysis of difference in algorithms' complexity when switching to the query ontology due to the fact that initial complexity evaluations are not available for all the tools reviewed here and the fact that profit from the query ontology use is depended on the complexity features of initial user ontology.

##### A. *OntoBuilder*

Information integration tool *OntoBuilder*, presented in [6], is the tool that is able to construct the ontology form the supplied HTML forms. It is also able to perform ontology schema matching if multiply forms were supplied.

Schema matching algorithm of *OntoBuilder* works as follows: first, it computes term and value similarity producing linguistic similarity value; then, it calculates composition similarity for each pair of terms using mean linguistic similarity value of term's siblings and parents; next, it evaluates precedence similarity for each pair of terms; finally, it outputs combined weighted value of term, value, composition and precedence similarities.

It's clear, that preparing query ontology one should consider the *OntoBuilder*'s heuristics of composition and precedence similarity. To evaluate correct composition similarity values for the terms of interest, we need to keep in query ontology all the siblings and parents of those elements. Respectively, to evaluate precedence similarity value, we need to preserve elements which are known to precede the elements of interest.

Moreover, since the *OntoBuilder*'s composition and precedence matching algorithm is not recursive, there is no need to evaluate the matching value for each element that appears in query ontology; therefore, it's enough to do the

matching process only for those elements, which appear in the user query. Due to this fact, the computation complexity of both composition and precedence algorithms will be reduced by the factor of at least  $n/m$ , where  $n$  is the size of global ontology and  $m$  is the number of all terms, preserved in user query.

### B. Cupid

System Cupid [5] was developed under Microsoft Research. Cupid is general schema matching algorithm, which includes atomic linguistic-based matching and exploits both element and structure information performing match based on local vicinity of matched elements.

Cupid analysis consists of three major steps: first, Cupid evaluates linguistic match between each pair of elements based on substring matching and auxiliary sources; then, it deduces structural similarity from elements' linguistic matches and local vicinity; finally, it outputs combined value of weighted linguistic and structure similarities.

Since query ontology introduces only structural changes to the full ontology, we are not interested in the linguistic matching algorithm here. Contrary, the structural matching algorithm of Cupid should be considered when constructing the query ontology.

The structure matching algorithm of Cupid is recursive and built in that way so the structural similarity of the non-leaf elements depend upon the fracture of strongly similar (defined by threshold value  $th_{accept}$ ) elements in their leaf sets. Respectively, the structural similarities of elements in the leaf set is increased every time when the combined linguistic-structural similarity of their parent element exceeds certain  $th_{high}$  threshold (or decreased if it is lower then certain  $th_{low}$ ). Moreover, to calculate non-leaf elements' matches, algorithm needs second run. In this way, as it is stated by authors in the [5] paper, the structure matching algorithm is biased towards leaf elements which ideally should bear most of the schema semantics.

Recursive nature of algorithm means there is no straightforward way for Cupid of reducing global ontology to the query ontology – each element affects the similarity value of all the other schema elements. However, we still can try to reduce the global ontology if exploit the fact Cupid is biased towards leaf elements. For example, we can save in query ontology all the leaf elements of elements of interest as well as all the elements along the path from the elements of interest up to the root element. This will affect the structural match values of the elements that were left in query ontology in two ways. First, non-leaf elements structural similarity value can be lowered if the fracture of strong links in the trashed leaf set of this element was higher then of those, preserved in query ontology; contrary, it will be higher if noted fraction was smaller; at last, it will remain the same if the fractions were the same. Second, for the leaf elements structure similarity can remain unadjusted if their ancestors' similarity didn't exceed either  $th_{high}$  or  $th_{low}$  threshold due to the fact, mentioned in the previous reason. Additionally, due to the recursive flavor of

the algorithm, if non-leaf element's structure similarity didn't exceed the thresholds, the leaf elements similarity will remain unchanged being under or above the  $th_{accept}$  threshold therefore affecting the structure similarity of this non-leaf element's ancestors.

To the large degree, aforementioned effects are caused by the system of threshold values used in the algorithm and, therefore, can be corrected adjusting these values. In the Cupid [5] paper, threshold values are referred to as typical and are chosen by human. We argue that those values can be calculated through the pre-analysis of the supplied ontology. For example, we can try to match ontology schema with itself; then,  $th_{accept}$  value can be raised for the elements that can be matched incorrectly and lowered for the terms that appear in ontology exactly once and therefore can't be matched incorrectly.

After all, the cumulative effect of the query ontology on the Cupid matching algorithm is hard to predict and will rely greatly on the characteristics of the initial global ontology (e.g. structure, number and position of elements that potentially can be matched incorrectly, etc.) as well as chosen threshold values.

### C. BYU

D. Embly et al. in [2] discuss the approach for extraction and structuring of unstructured data from data-rich WEB-documents. First, tool splits document into records (for example, in car classifieds page each record will represent one car advertisement). Record splitting is done with the help of five heuristics ([11]): highest-count tags, identifiable "separator" tags, standard deviation, repeating-tag pattern, and ontology matching. Ontology matching is done using so-called "record-identifying" fields of ontology, which are the fields that occurs exactly once in the record. For example, for the car ads make and model can serve as such the fields; contrary, car features can not. To work correctly, algorithm requires at least 3 record-identifying fields and no more then 20% of all available ontology objects.

Next, having defined records of data, tool calculates all possible candidates for the all the terms of given ontology and then tries to deduce the correct mapping from the candidates set using sort of ontology schema matching. Noted deduction is also done using the set of heuristics which includes the following: singleton heuristics; functional-group heuristics; nested-group heuristics; and precedence heuristic, which is not described in the paper as the separate heuristic, but still is assumed in the analysis process.

As it follows from the algorithm, the system is highly sensible to the quality of the supplied ontology: it requires good set of record-identifying fields as well as defined properties of precedence and group functionality over the basic ontology information. Nonetheless, these constrains can still be satisfied for the well-specified domains.

Generally, BYU algorithm performs all the operations over the terms only in the term's local context. However, there are two algorithm steps that require the global: both record

boundary discovery and singleton heuristic require all the ontology singletons. This properties force to keep in the ontology next entities: all the singletons; terms of interest; terms that appear in functional group of terms of interest; term that are the parents of the terms of interest (for the nested-group heuristic).

In this way, the ontology can be reduced just to the terms of interest, their context, and singleton terms. Due to the property of singleness of singleton terms appearance in the page, overall computation complexity should not increase as radically as if we were keeping all leaf terms. Further, with additional effort we can reduce the kept number of singletons since the record boundary discovery algorithm requires only the best 20% of the singletons and singleton heuristic will need only the singletons in the local context of terms of interest if this heuristic is applied after functional- and nested- group heuristics.

#### D. Pick-Up

Pick-Up[4] is a semi-structured data extraction (wrapping) tool that is able to iterated through the rows of the HTML page detecting table-like structures and thus extract tables from the page into the manipulateable data structure, which can be used later in combination with ontology to populate the desired database; moreover, Pick-up is also able to support nested tables which, in turn, can be composed of the WEB-pages, linked to the one under analysis.

Before any analysis, Pick-Up translates the required WEB-page into the “tag-tree” data model which flattens the page’s DOM describing each element by its path from the root element (you can think of fully qualified file name in the file systems). In the core of Pick-Up system lays the assumption that record structures in the document share structural pattern. Having these in mind, Pick-Up analyses the supplied document and identifies the repeated pattern (table row) based on its repeated elements (row cells). In order to fight a selection of wrong candidate tables, system is able to ask user to specify the starting element for analysis in the page. Then, if detected table cells contain the nested tables (either specified directly in text or supplied as hyperlinks), Pick-Up treats these nested tables as the separate documents and performs on them the same analysis as on the main document. Plus, user is able to specify the nesting for each column.

While being rather flexible system, Pick-Up doesn’t utilize the ontology information thus leaving schema matching algorithm to put constrains on the query ontology. Still, if used in autonomous integration system, Pick-Up performance can be optimized using query ontology. Since algorithm uses optimal pattern discovery process, it involves a lot of computation. However, in documents with complex nested structure user may probably need only the small fraction of available information. In this case, root element of the query ontology (see first consideration of subchapter 3.1. for additional explanations) can be located in the document using standard IR techniques and then can be supplied to the Pick-Up algorithm as the starting element for the analysis (see

above).

Next, query ontology can be used to determine the depth of nesting, i.e. system should expand only those columns, whose corresponding ontology element is not in the ontology leaf set.

In this way, with the help of query ontology Pick-Up extraction process can be turned from semi-autonomous to the fully autonomous and thus can be used in autonomous WEB data integration systems.

#### E. Discussion

Here, we demonstrated the sample algorithm analysis, which should be used while building the query ontology-based information system from different logical blocks such as schema matchers and data extraction tools. Moreover, to build the complete information system, query ontology should be constructed taking into account and combining the features of all underlying ontology-based subsystems. For example, Pick-Up system poses no restrictions on the used ontology structure, and, therefore can be used easily with the query ontology, built for the OntoBuilder; contrary, in case of BYU and Cupid tools combination, singleton heuristic is heavily used in BYU tool and, therefore, should be supported in resulting query ontology even though singletons are not used Cupid.

From the proposed examples, it is also clear that query ontology is better suitable for the ontology-based systems, which are not recursive and therefore are based only on the limited term’s local context. Further, in the latter case it’s usually possible to reduce the algorithms’ computation complexity just to the preserved context of elements of interest.

## V. RELATED WORK

Query ontology concept is sometimes used interchangeably (mistakenly to authors opinion) with the user ontology concept. User ontology concept refers to the GAV (global-as-view) and LAV (local-as-view) models of user query processing in global information systems [12]. Accordingly to this model, user ontology represents model of user (information seeker) knowledge whereas domain ontologies describes the knowledge model of information providers. While our approach employs similar GAV model, our query ontology is only the small part of the whole user (or global – which ) ontology, constructed using user query and knowledge about employed information systems algorithms.

E.Mena et al. in [13] combines User Ontology with user query by encoding the query as Ontology constrains and projections. The information loss that occurs during translations between ontologies is also discussed. However, the user ontology’s structure and terms in this approach remain unchanged; moreover, it combines the user ontology with desired domain ontology if the user is not satisfied with the query result.

Query Ontology concept similar to the one presented here was used in [14]. System proposes to construct light-weight ontology from the global ontology by preserving all the superclasses along the path from the elements of interest up to

the root of the ontology; also, all the attributes of preserved superclasses are preserved as the attributes of elements of interest. Nonetheless, the paper presents no reasoning for this. Plus, the proposed system used query ontology for different task – information seeking and communication in agent-based systems.

## VI. CONCLUSIONS AND FUTURE WORK

In this paper we presented query ontology, the improvement concept over the existing WEB data integration systems, which is bale to optimize the whole integration process. The proposed architecture can bring up couple positive sides to the global information systems. First, it can greatly reduce the amount of data transferred trough the system; this, in turn, allows reducing of computation operations and increases overall system throughput. Secondly, as it was shown on the example of Pick-Up tool, query ontology can help to automate the tools, which typically requires certain human information when additional knowledge is needed. On the down side, in certain cases, as it was shown on the example of the Cupid system, loss of information caused by global ontology truncation can decrease the overall quality of the information, provided by the system, and make the system unstable; still, as it is indicated in [15] for example, in certain cases decreased quality of the provided information can be acceptable if the user is more concerned about the quickness of the system's response rather than quality.

Further, even though the proposed approach promises to reduce greatly the computational complexity of ontology-based information systems' operations, it would be incorrect to think that proposed approach can bring the full process of WEB data integration to the users' desktops; availability of this technology is buried under the bottleneck of cumulative Internet bandwidth. The preliminary feasibility estimations can be found in [16] for example and state that current Internet is incapable of delivering the required amounts of data to the user's computer. Instead, we envision the use of proposed framework for two purposes: in the first case, user can take the advantage of reduced computation cost for integration of relatively small amounts of data on his own average-performance workstation; in the second, more interesting, case, huge Internet services providers like Google[17] can provide integration service of pre-cached data. It is clear that in the latter case it will be definitely possible to make profit of efficient computations in the face of massive loads that such the services usually experience.

Finally, we have the intuition that appropriately designed query ontology can also be used to enrich regular Internet search queries. In this case, terms from the query ontology can be added to the search query to restrict it and therefore narrow the scope of the search. Also, there exist opposite side of user query data enrichment – the autonomous integration and search system can offer user additional items, that can be related to his query. Though in this area exists certain work, conducted by other researchers, we believe that these

statements are still questionable and needs further investigation in the context of specific search and/or integration engine.

## REFERENCES

- [1] R. Hull, "Managing Semantic Heterogeneity in Databases: A Theoretical Perspective," presented at Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, 1997.
- [2] D. W. Embley, D. M. Campbell, R. D. Smith, and S. W. Liddle, "Ontology-Based Extraction and Structuring of Information from Data-Rich Unstructured Documents," presented at Conference on Information and Knowledge Management, 1998.
- [3] T. R. Gruber, "A Translation Approach to Portable Ontology Specifications," *Knowledge Acquisition*, vol. 5, pp. 199-220, 1993.
- [4] L. Chen and H. M. Jamil, "On using remote user defined functions as wrappers for biological database interoperability," *International Journal on Cooperative Information Systems*, 2003.
- [5] P. A. Bernstein, J. Madhavan, and E. Rahm, "Generic Schema Matching with Cupid," presented at VLDB Conference, 2001.
- [6] G. M. A. Gal, H. Jamil, "Ontobuilder: Fully automatic extraction and consolidation of ontologies from web sources," presented at IEEE International Conference on Data Engineering, 2004.
- [7] M. Bunge, *Treatise on Basic Philosophy: Volume 3: Ontology I: The Furniture of the World*. Reidel, Dordrecht, Holland, 1977.
- [8] M. Bunge, *Treatise on Basic Philosophy: Volume 4: Ontology II: A World of Systems*. Reidel, Dordrecht, Holland, 1977.
- [9] A. Laender, B. Ribeiro-Neto, A. Silva, and J. Teixeira, "A Brief Survey of Web Data Extraction Tools," *SIGMOD Record*, vol. 31, pp. 84 - 93, 2002.
- [10] E. Rahm and P. A. Bernstein, "A survey of approaches to automatic schema matching," *The International Journal on Very Large Data Bases*, vol. 10, pp. 334–350, 2001.
- [11] D. Embley, S. Jiang, and Y. Ng, "Record-Boundary Discovery in Web Documents," presented at ACM SIGMOD International Conference on Management of Data, 1999.
- [12] E. Mena and A. Illarramendi, *Ontology-Based Query Processing for Global Information*, vol. 619. Norwell, MA, USA: Kluwer Academic Publishers, 2001.
- [13] E. Mena, V. Kashyap, A. Illarramendi, and A. Sheth, "Imprecise Answers In Distributed Environments: Estimation Of Information Loss For Multi-Ontology Based Query Processing," *International Journal of Cooperative Information Systems*, vol. 9, pp. 403--425, 2000.
- [14] D. R. Antonio Moreno, David Isern, Jaime Bocio, David Sánchez, "Agent-based semantic information search on the Web," presented at International Workshop on

Practical Applications of Agents and Multi-Agent Systems, 2004.

- [15] A. Silberschatz and S. Zdonik, "Database systems—breaking out of the box," *ACM SIGMOD Record*, vol. 26, pp. 36 - 50, 1997.

- [16] Jinyang Li, Boon Thau Loo, Joseph M. Hellerstein, M. Frans Kaashoek, David R. Karger, and R. Morris, "On the Feasibility of Peer-to-Peer Web Indexing and Search," presented at Second International Workshop on Peer-to-Peer Systems, 2003.

- [17] Google, "Google."