Bio-Medical Data Integration Based on MetaQuerier Architecture

[1]Khondker Shajadul Hasan, [2]Munirul Islam, [3]M Samiullah Chowdhury, [3]Eusuf Abdullah Mim, and [3]Naieem Khan

[1]School of Computer Science, University of Oklahoma, 110 W. Boyd St., Norman, OK 73019, USA
shajadul@ou.edu

[2]Department of Computer Science, Wayne State University, Detroit, MI 48202, USA
munirul@wayne.edu

[3]Department of EECS, North South University, Bashundhara, Dhaka, Bangladesh, {samiullah.chowdhury, eusuf9001, ntkhan}@yahoo.com

Abstract - *The emergence of a large number of bio medical datasets on the Internet has resulted in the need for flexible and efficient approaches to integrate information from multiple bio medical data sources and services. Thus data are scattered in different web sites and web databases. User struggling hard and for them it is extremely difficult for them to find accurate data from the web efficiently. In this paper, we tried to present our approach to establish an architecture which will automatically generate web data integration, optimize the composition, and execute the required output efficiently. While data integration techniques have been applied to the bio medical data domain, the focus has been on answering specific user queries. Thus we have found the indication towards large scale data integration. So the issue arises for which data integration architecture can be used. There are so many proposed large scale data integration architecture are available. Among all of them we designed our paper based on the MetaQuerier architecture. It's large scale integration over web databases. MetaQuerier architecture has five basic processes which will be clarified in this paper briefly. We used this architecture to implement our bio medical data integration and try to generate a well structured output. Here our first task is to explore the MetaQuerier architecture and secondly we will explore the design in terms of bio medical data.*

Keywords: *MetaQuerier architecture, Data crawling, Source clustering, Schema etc.*

I. INTRODUCTION

Biologists are now faced with the problem of integrating information from multiple heterogeneous public sources with their own experimental data contained in individual sources. The selection of the sources to be considered is thus critically important. There is a compelling demand for the integration and exploitation of heterogeneous biomedical information for improved clinical practice, medical research, and personalized healthcare across the EU. The ultimate goal of the project is to provide uninhibited access to universal biomedical knowledge repositories, large-scale information-based biomedical research and training.

Now-a-days new treatments come about as a result of other, earlier discoveries. They are often unconnected to each other, and in various field. Sometimes the research was done for non-medical purpose and only by accident contributes to the field of medicine. Like the discoveries of *penicillin.* But now all the treatment has to be done through research. In the terms of Bio-Medical, we are considering the data from medical diseases , different kind of elements of human being and analysis of various medicine, the experiments and result obtained from them, analyzing zinc, proteins, bacteria and many more for advance research. For example, changes in genomic DNA, presence of various protein modification, mRNA and protein levels etc. The possibilities from bio-medical data integration are enormous. For example, the central tumor suppressor protein p53 provides a potential target for new anti-cancer drugs. By integrating the datasets from different laboratory various result of protein p53 like its characteristic, behavior, effect, mutations, etc. in one single database.

Data for bio-medical researches integrated from the web. Data can be stored on the WEB in different form. The data can be non-structured or semi structured in the web. Like plain text files, HTML text files, native XML. Data might be found in online libraries, catalogues, etc. Databases in the research repositories are like genome databases, scientific databases, environmental databases, etc. There might be web services, semantic web, and knowledge base system. There are Ontologies, which are structurally and semantically research domain description with associated data. The following charts are important for our paper. Here we have shown some biomedical data sets. There are unlimited bio medical datasets. Here our main focus actually to introduce the fact that bio medical data are needed to be integrated and also it is possible to be integrated. So in order to clarify our paper we focus on specific data sets which are in the group of Protein.

Database systems except from the web must have to inter-operate, cooperate and coordinate with each other. These data have to shared, exchanged and ultimately integrated. In this regard our target to see the sights of the

Table 1: Contains an example of Bio-medical data and attributes. [2]

Concept	Source
Protein	HSProtein($id, name, location, function, sequence, pubmedid)
	MMProtein($id, name, location, function, sequence, pubmedid)
	MembraneProtein($id, name, taxonid, function, sequence, pubmedid)
	TransducerProtein($id, name, taxonid, location, sequence, pubmedid)
	DIPProtein($id, name, function, location, taxonid)
	ProteinLocations($id, $name, location)
Protein -Protein Interactions	HSProteinInteractions($fromid, toid, source, verified)
	MMProteinInteractions($fromid, toid, source, verified)

MetaQuerier architecture. MetaQuerier architecture is one of the most recent data integration architecture. The following figure will give some basic idea about the data flow and data manipulation inside the MetaQuerier mechanism.

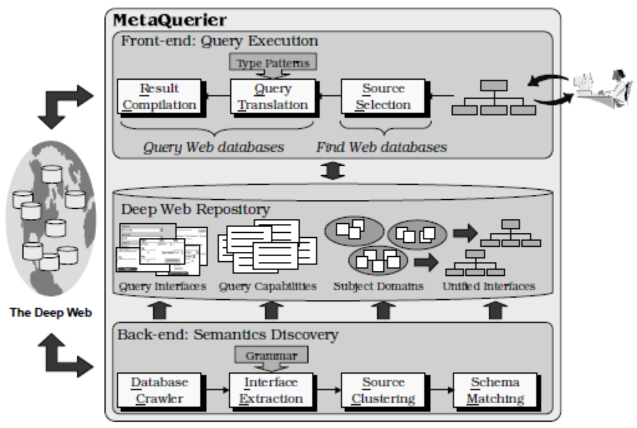


Figure 1: MetaQuerier: System Architecture. [1]

The above figure 1 is the complete scenario of the MetaQuerier architecture. Based on the MetaQuerier mechanism, our goal has two directions– First, to make the deep Web systematically accessible, it will help users find online databases useful for their queries. Second, to make the deep Web uniformly usable, it will help users query online databases [1]. In this paper we first describe the work flow of MetaQuerier engine and then explore the bio medical data inside MetaQuerier. At the end, Integration of related works will clarify the necessity of MetaQuerier in bio medical data integration. Actually the necessity of an efficient data integration engine arises due to the extremely huge volume of queryable databases. One side the data collection are dynamic another side they are non systemic.

To our knowledge, our goal of integration at a large scale has largely remained unexplored. The MetaQuerier engine actually integrates data from the web. One of the critical issues is that data are not predefined. Data are flourishing in every moment and datasets are getting larger. Since datasets are not predefined data discovery become dynamic. If one user search for different types of protein for example, for the next search he or she will not get the same datasets from the web resources. So this is a challenge while data searching. That’s why we need data crawler. Another major issue which steps in more complexity situation is that data are needed to be integrated on the fly. The engine will work at a time [1].

II. Related Work

Our complete work has two basic direction and stands. One part deals with MetaQuerier engine and other part focus on bio medical data and how MetaQuerier engine will manipulate bio medical data. What do we understand about bio medical data? There exist a large number of bio medical datasets on the web in various formats. There is a need for flexible and efficient approaches to integrate information from these datasets. Unlike other domains, the bio medical domain has hold web standards, such as XML and web services. There exists a large number of bio medical data sources that are either accessible as web services or provide data using XML. For the bio medical data sources that provide their data as semi-structured web or text documents, we can use wrapper- based techniques to access the data.

For example, when a user queries the UniProt1 website for details of a protein, the user provides a uniprotid and gets back the information about the protein. The emergence of the large number of information providing services has highlighted the need for a framework to integrate information from the available data sources and services. In this paper, we describe our approach to automatically compose integration procedure to create new information-providing the MetaQuerier engine.

When the MetaQuerier receives a request based on bio medical data to create a new web service, it generates a parameterized integration that accepts the values of the input parameters such as protein name or its id and then retrieves and integrates information from relevant web pages, and returns the results to the user. The parameterized integration procedure is then hosted as a new web page what is known as data crawling. The discoveries of the pages according to the user requirement are dynamic and they are absolutely unsorted. This is the key challenge in composing web data for a new web based on the fly integration.

To further clarify these consider the example shown in Figure 2. We have access to three web services where each providing protein information for different organisms. We would like to create a new web service that accepts the name of an organism and the id of a protein and returns the protein information from the relevant web service. Given specific values of the input parameters, traditional data integration systems can decide which web service should be queried. However, without knowing the values of the parameters, the traditional integration systems would generate a procedure that requires querying all three web services for each request.

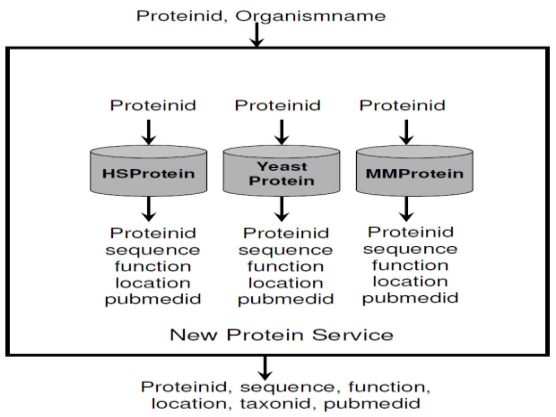

Proteinid, Organismname
Proteinid
Proteinid
Proteinid
HSProtein
Yeast Protein
MMProtein
Proteinid sequence function location pubmedid
Proteinid sequence function location pubmedid
Proteinid sequence function location pubmedid
New Protein Service
Proteinid, sequence, function, location, taxonid, pubmedid

Figure 2: Protein Information. [2]

The key contribution of our approach is to extend the existing techniques to generate parameterized integration technique that can answer requests with different sets of values for the input parameters.

Now the key issue arises when it's needed to optimize the web data and in order to reduce the deep web data for optimizing the number of user request sent to existing data sources we need the help of MetaQuerier. Thus when data optimization with user satisfaction will be needed MetaQuerier will be in action. The existing optimization techniques means the MetaQuerier will utilize the searchable user query to filter out unnecessary source requests and/or reorder the joins to produce more efficient ordered web data. However, as we show with a detailed example later in the paper, the MetaQuerier techniques are enough when we apply them to the task of data integration.

Next section will describe each of the processes of the MetaQuerier techniques in more detail, show how they can be applied to the bio medical domain. We begin by describing a motivating example that we use throughout the paper to provide a detailed explanation of various concepts. Next, we discuss how existing data integration techniques can be extended to model web sources as data sources and reformulate web data creation requests into parameterized integrated data.

In MetaQuerier there are five basic processes. First is Database crawler, second Interface Extraction, third source clustering, fourth Schema Matching and fifth Result Compilation. [1]

There are three parts of the complete MetaQuerier. Front end, back end and deep web repository. But before understand the parts of MetaQuerier we need to understand its starting and action point. As it handles large volume of data, first, such integration is dynamic: Since sources are blooming and evolving on the Web, they cannot be statically configured for integration. Second, it is absolutely unsorted: Since queries are submitted by users for different needs, they will each interact with different sources. Thus, toward the large-scale integration, the MetaQuerier must achieve dual requirements–Dynamics discovery and on the fly integration. To our knowledge, MetaQuerier is the first one to present the overall system issues of building large scale integration. Next section we will elaborate about MetaQuerier architecture. [1]

III. System Descriptions

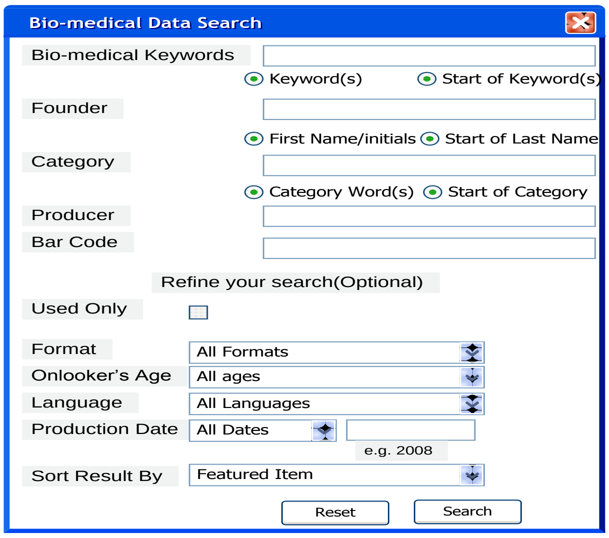
On the way towards bio-medical data integration, we have accounted the large scale of data and these data can be found on the web database. But data are not predefined. It means, we are doing a deep web searching as user's request but sources are not in a single domain, which we are calling "dynamic discoveries'" and "on-the-fly" integration. Based on the processes used in MetaQuerier, we have structured the idea of bio-medical data integration. We will now describe the whole system of the application. [1]

The MetaQuerier was developed for large scale integration. In its way of integration, it basically search and collect the database on the web, extract the required data from the database and gather it into its own database and show users the output as requested. To understand the system easily, we have divided the whole process into five major parts. On sequence, Data Crawling, Interface extraction, Source clustering, Schema Matching, Query Translation, Source Selection, and Result Compilation. Data Crawling, Interface extraction, Source Clustering, Schema Matching all these processes work at the back-end. Query Translation, Source selection and Result Compilation all these work on front-end. In this interim paper we are giving a short brief for ease of understand bio-medical data integration. [1]

A. *Data Crawling*

Collect data from enormous web environment is the main part of the challenge that we face while data integration. So actually we need data crawler. There is a difference between data crawler and web crawler. Existing and available search engines are efficient for necessary site searching. They search based on the root pages and also check user keywords as interface keyword [3]. Here if we go in that process we will be in the messy situation of managing terabyte of data. So in the MetaQuerier our task to find web pages that are exclusively important for us including the databases involved with these sites. Thus MetaQuerier design data crawling in two different segments to face the challenge of dynamic discovery. The first segment named site crawler and second segment is shallow crawler [1]. Together these two segments MetaQuerier named the data crawler as site based crawler. For site crawler the efficiency of query interfaces are very important.

Query interfaces are important because based on the interface keyword crawler will filter web sites. It will minimize unsorted and unnecessary data. Suppose the following interface can use to find more appropriate and mandatory data while search wed sites. The more keyword will be used from the interface the more data filtering will be in action. Since we are not focusing on how efficient interface can be designed here, we just discuss the important of query interfaces and its necessity for data integration. In this paper we are actually trying to explore one of the important uses of data integration in the field of bio medical research. Site base crawler will go through the root pages and will indentify IP addresses and shallow crawler will follow these IP addresses and will search web servers which will be found from site crawler [1].


Bio-medical Data Search

Bio-medical Keywords	
	Keyword(s) / Start of Keyword(s)
Founder	
	First Name/initials / Start of Last Name
Category	
	Category Word(s) / Start of Category
Producer	
Bar Code	
Refine your search(Optional)	
Used Only	☐
Format	All Formats
Onlooker's Age	All ages
Language	All Languages
Production Date	All Dates (e.g. 2008)
Sort Result By	Featured Item

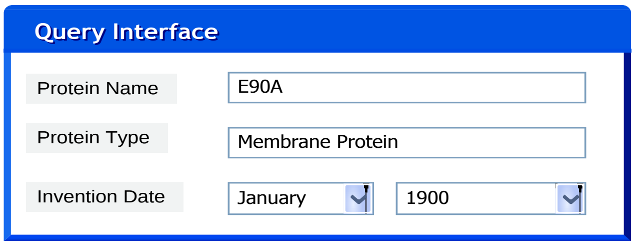
Figure 3: Sample Search Interface. [9]

B. *Interface Extraction*

Interface extraction basically extracts the required data from the query interfaces. Query interface sometimes share similar or common query patterns but sometimes it shares different query patterns. In case of different query patterns the problem arises due to some hidden information or attributes. Hidden attributes are not visual on interface that's why its extraction normally out of interaction at the beginning. Thus for the hypothesized syntax, in metaQuierer the determined structure are rationalize by asserting the creation of query interfaces as guided by some hypothetical syntax [5]. Therefore handle this hypothetical syntax effectively creates new problem. So it's needed to be visualizing as a visual language whose composition conforms to a hidden non-prescribed, grammar. In this case MetaQuerier solve the problem in terms of parsing the visual language. Here the MetaQuerier approach is to introduce a parsing paradigm by assuming that there exists hidden syntax to describe the layout and semantic of query interfaces. Specifically, we develop the subsystem IE as a visual language parser, given a query interface in HTML format; IE tokenizes the page, parses the tokens, and then merges potentially multiple parse trees, to finally generate the query capability. [1,4]

Finally after parsing Interface Extraction basically extracts query capabilities from the query interfaces. The semantically related labels and elements of a search interface are viewed as logical attributes, though they are scattered in the html text or into the database without formal definitions. Therefore, attributes have to be identified by grouping associated labels and elements. Moreover, beyond the labels and elements, a significant amount of semantic/meta information for attributes exists on the query interfaces [5].

For example, in figure 4, "invention date" implies the Attribute is semantically a date data type, and its two elements are used to specify a range query condition with different roles in specifying the condition. Unlike the conventional database schemas, such semantic/meta information is "hidden" from computers and not formally defined on query interfaces. As such, the "hidden" information about each attribute needs to be revealed and defined to enrich the schema matching. [5]



Query Interface

Protein Name	E90A
Protein Type	Membrane Protein
Invention Date	January / 1900

Figure 4: Sample query interface.

C. *Source Clustering*

Before move on to Schema Matching we need to understand Source Clustering. Source clustering collaborates with source selection which works in front end. These two processes help schema matching to get actual scenario. After determining query capabilities based on query interfaces source clustering sorted data as mediated process which provides data towards schema process. Here the second challenge of MetaQuerier after the dynamic discovery, the on the fly integration comes in action. Source Cluster actually clusters sources according to subject domain. Going towards data integration, we need clustering sources by their query capabilities, specifically, given a set of query capabilities representing structured sources, our task is thus to construct a hierarchy of clusters, each representing an object domain of "structurally-consistent" sources. Thus we need to cluster the query interfaces into subject domains.

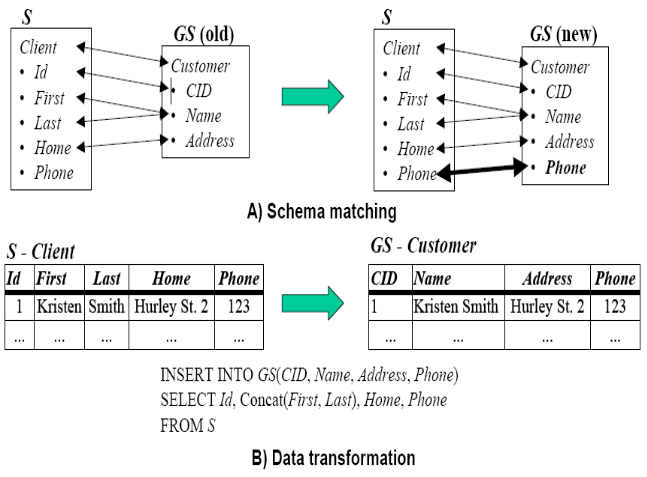
Domain elements and constraint elements have the following characteristics:

- Textboxes cannot be used for constraint elements.
- Radio buttons or checkboxes or selection lists may appear as constraint elements.
- An attribute consists of a single element cannot have constraint elements.
- An attribute consisting of only radio buttons or checkboxes does not have constraint elements.

Based on these characteristics, a simple two-step method has to be used to differentiate domain elements and constraint elements. First of all, we have to identify the attributes that contain only one element or whose elements are all radio buttons, or checkboxes or textboxes. Such attributes are considered to have only domain elements. Then an Element Classifier is needed to process other attributes that may contain both domain elements and constraint elements. [1,9]

D. *Schema Matching*

The schema of a database system is its structure described in a formal language supported by the database management system. In a relational database, the schema defines the tables, the fields in each table, and the relationships between fields and tables. Schemas are generally stored in a data dictionary. Although a schema is defined in text database language, the term is often used to refer to a graphical depiction of the database structure. Schema matching is the process of identifying that two objects are semantically related while mapping refers to the transformations between the objects.



S - Client

Id	*First*	*Last*	*Home*	*Phone*
1	Kristen	Smith	Hurley St. 2	123
...	...	...	...	...

GS - Customer

CID	*Name*	*Address*	*Phone*
1	Kristen Smith	Hurley St. 2	123
...	...	...	...

```
INSERT INTO GS(CID, Name, Address, Phone)
SELECT Id, Concat(First, Last), Home, Phone
FROM S
```

Figure 5: Schema matching for data integration.

In data integration process, schema matching find out the semantic domain values among the attributes, which we have found through query interfaces. In MetaQuerier, the subsystem "Schema matching" was developed for the large scale databases, which stored the discovered matching elements in the deep web repository. Schema Matching is an important process which use data from **query capabilities** and organize the data as per requirement. It provides data to **source selection** and **Query Translation** has the urge such data for translate user's query and finally send it to users at the front-end. What MetaQuerier actually does, it redesigns the process in terms of complex matching instead of one by one process. [7]

E. Query Translation

Query Translation is a front-end process. We know that with all those massive sources, the deep Web is clearly an important frontier for data integration. In particular, to enable query mediation for effective access of Web databases, it is critical to automatically translate queries across their **query interfaces**. Such translation is, in essence, to match and express query conditions in terms of what an interface can "say": Each query interface consists of a set of constraint templates. For complete query translation, first we need to extract constraint templates from a **query interface**. Second, from given source and target constraint templates, we need to find matching templates. Specifically, the deep Web is of large scale and of a dynamic nature as the sources are changing and new ones are emerging. Also, it is very diverse, with various sources. Users will thus interact with "ad-hoc" or unplanned sources to satisfy their various information needs. This large-scale, dynamic, and unplanned nature mandates effective integration to enable "on the fly" query translation. That is, the mapping technique should be able to translate queries for unseen sources, where no pre-configured translation knowledge can be assumed. In the MetaQuerier, query translation will translate the user's query to query interface. [6]

F. Source Selection

From the large scale database, following the typical approach to data integration we define a common mediated schema for all the data sources, then to match and map the data sources to this mediated schema [1]. The target user may understand the concepts in their own domain, means they know how and where to search, but may not know the data on other domains. In case of this problem, we need to choose which sources to include in the data integration and what mediated schema to use. So our goal is to develop "source selection" subsystem to choose a set of data sources and a global mediated schema over these resources.

G. Result Compilation

This is the final process of data integration which essentially aggregates query results to the user. Result Compilation compiles data results from different sources into coherent pieces. For establishing result compilation, we need to build it for extracting data from schema matching and matching other attributes across different sources.[1]

IV. PROCESSES WITH BIO MEDICAL DATA

In this section we describe an extension to the existing data integration techniques to solve the problem of generating parameterized integration plan for new bio medical web services. Most Life Sciences web services are information-providing services. We can treat information-providing services as data sources with binding restrictions. Data integration systems require a set of domain relations, a set of source relations, and a set of rules that define the relationships between the source relations and the domain relations.

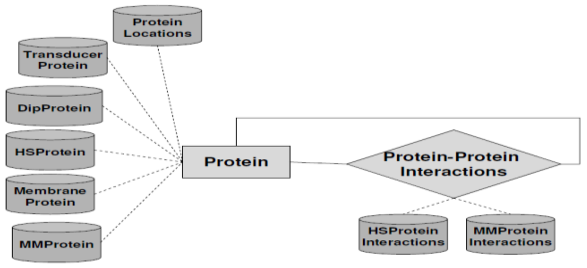


Figure 6: Relationships between Domain Concepts and Data Sources. [2]

In order to utilize the existing web services as data sources, we need to model them as available data sources and create rules to relate the existing web services with various concepts in the domain. Typically, a domain expert consults the users and determines a set of domain relations. The users form their queries on the domain relations. We have two domain relations with the following attributes:

- Protein (id, name, location, function, sequence, pubmedid, taxonid)
- ProteinProteinInteractions (fromid, toid, taxonid, source, verified) [2].

The Protein relation provides information about different proteins. The Protein-Protein Interactions relation contains interactions between different proteins. As the id attribute in the Protein relation is the primary key, all other attributes in the Protein relation functionally depend on the id attribute. For the Protein Protein Interactions domain relation, the combination of fromid and toid forms a primary key. [2]

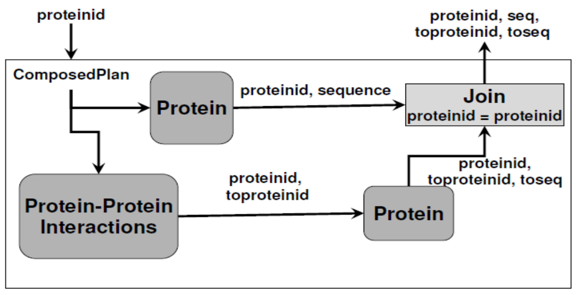


Figure 7: Integration Plan of a Desired Web Service [2]

A. Database Crawler

On a search interface, an attribute may have multiple associated elements and they may be related in different ways. There can exist four types of element relationships: *range type*, *part type*, *group type* (multiple checkboxes/radio buttons are sometimes used together to form a single semantic concept/attribute) and *constraint type*. For example, in Figure 8, the relationships between the elements of Production Year are of *range type.*

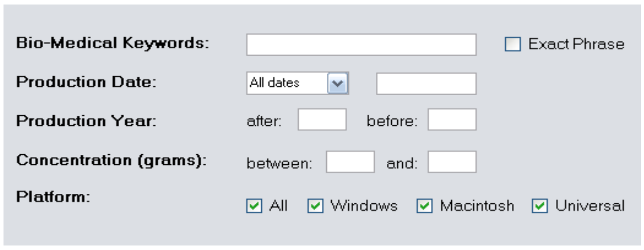


Figure 8: Example of element relations.

When an attribute has multiple associated elements, we shall classify them into two types: *domain elements* and *constraint elements* because they usually play different roles in specifying a query. Domain elements are used to specify domain values for the attribute while constraint elements enforce some constraints to domain elements. For example, element "Exact phrase" is a constraint element while the textbox following Title keywords is a domain element.

Consider two interfaces. One interface contains an attribute protein Production date, and another interface contains an attribute protein production year, and they should be matched in terms of their semantics. But we cannot match them by only using names because they do not have exactly the same attribute name. However, if we can identify that the elements of both attributes are of *range type*, it would increase the confidence of matching them. When a user specifies a query on the global attribute Title of a MetaQuerier interface, during the query translation the query value should be mapped to the domain element of Bio-Medical keywords instead of the constraint element "Exact phrase". [3]

B. *Interface Extraction*

Labels and elements are the basic components of a search interface, but it is insufficient to just extract individual labels and elements because many applications rely on the *logical attributes* formed by related labels and elements. In order to extract logical attributes, it is essential to determine the semantic associations of labels and elements. However, there are no explicit definitions of such associations in the HTML text of the search interface. We observe that labels and elements that represent the same attribute have a certain layout pattern and are usually close to each other and that in most cases they have some similar information in common. On the basis of this, we develop a three-step approach to tackle the problem of automatic interface extraction or in other words attribute extraction.

1) *Extracting Individual Labels and Elements*

This is the first step of our automatic attribute extraction method. Given a search interface, the extraction starts with its "<FORM>" tag. Each element itself contains its values (if available). Four types of input elements are considered: *textbox*, *selection list*, *and checkbox* and *radio button*. When a row delimiter like "<BR>", "<P>" or "</TR>" is encountered, a '|' is appended to the Interface expression. This process continues until the "</FORM>" tag is encountered. In this process, some irrelevant texts may be included in the INTERFACE EXPRESSION even though some efforts are made to identify and discard them. [8]

2) *Identifying the Names of Exclusive Attributes*

Exclusive attributes are actually the ones whose names may appear as *values* in some elements, such as a group of *radio buttons* or a *selection list*. Correctly recognizing such attributes automatically is difficult because they do not appear on search interfaces as descriptive texts. [5]

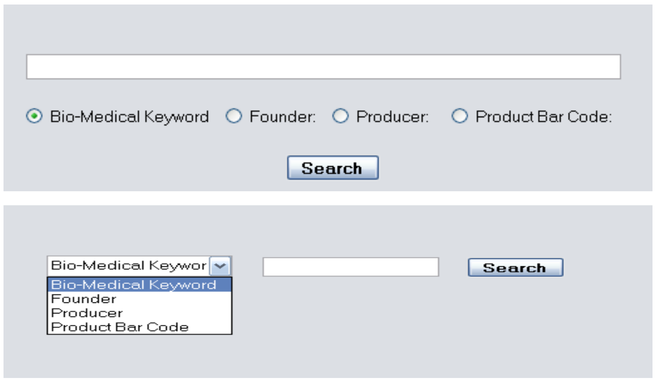


Figure 9: Examples of exclusive attributes. [9]

Exclusive attributes appear frequently on real Web search interfaces. A significant flaw of existing approaches for interface extraction is that they do not extract exclusive attributes.

The names of exclusive attributes are often the *most commonly used attribute names* of a domain. The basic idea is that we consider multiple interfaces in the same domain at the same time rather than separately. Then we use the extracted labels from all search interfaces of the same domain to construct a vocabulary for the domain. Finally we use the vocabulary to automatically identify and extract the names of exclusive attributes.

3) *Grouping Labels and Elements*

This step is to group the labels and elements that semantically correspond to the same attribute, and to find the appropriate attribute label/name for each group. For example, label "Bio-Medical Keywords", the textbox, the three radio buttons and their values below the textbox all belong to the same attribute and this step aims to group them together and identify label "Bio-Medical Keywords" as the name of the attribute.

C. *Source Clustering*

Going towards MetaQuerier, we need clustering sources by their *query schemas*, i.e., attributes in their query interfaces.

Table 2: Translation Rules.

r_1	[category; *contain*; \$s] → *emit:* [source; *all*; \$s]
r_2	[name; *contain*; \$t] → *emit:* [name; *contain*; \$t]
r_3	[concentration range; *between*; \$s, \$t] → \$p = ChooseClosestNum(\$s), *emit:* [concentration; *less than*; \$p]
r_4	[onlooker's age; *between*; \$s] → \$r = ChooseClosestRange(\$s), *emit:* [age; *between*; \$r]

For instance, for the advanced query interface of amazon.com, the query schema is specifically, given a set of query schemas representing structured sources, our task is thus to construct a *hierarchy* of clusters, each representing an object domain of "structurally-homogeneous" sources [1]. Apparently, we are focusing on Bio-Medical data. We explain a particular method of source clustering, which is quite efficient in terms of domain attributes.

1) Deriving information from Attributes

In our proposed interface schema model, we recommend four types of information for each attribute are defined: *domain type*, *value type*, *default value* and *unit*. These meta-data are only for *domain elements* of each attribute.

Domain type: Domain type indicates how many distinct values can be used for an attribute for queries. Four domain types are defined in our model: *range*, *infinite* and *Boolean*. [9]

Value type: Each attribute on a search interface has its own semantic value type even though all input values are treated as text values to be sent to Web databases through HTTP. [9]

Default value: Default values in many cases indicate some semantics of the attributes. A default value may occur in a selection list, a group of radio buttons and a group of checkboxes. It is always marked as "checked" or "selected" in the HTML text of search forms. Therefore, it is easy to identify default values. [9]

Unit: A unit defines the meaning of an attribute value (e.g., *kilogram* is a unit for *weight*). Different sites may use different units for values of the same attributes. For example, one search interface may use "Milligrams" as the unit of its Concentration attribute, while another may use "Liters" for its Concentration attribute. [9]

2) Translation Rules

Firstly, we have to consider another term named, query mediation. Query mediation works have been mainly focusing on mediating queries across multiple sources and thus abstract the problem as a paradigm of answering query using views. In particular, they assume each source has a wrapper, which encapsulates the tasks of extracting query capability, schema matching and constraint mapping for that source. The main focus of query mediation is thus on how to decompose a user query into sub-queries across multiple sources. In contrast, we have to focus on query translation between two sources other than mediating queries across multiple sources. In particular, we are dealing with the mapping of constraint heterogeneity. For our scenario of large scale integration, we have to on-the-fly translated queries and thus need the following mapping techniques.

Secondly, Schema mapping aims at translating a set of data values from one source to another one, according to given matching. Therefore, schema mapping only concerns about the equality relation between different schemas, based upon which data is converted. In particular, no constraint heterogeneity is considered in schema mapping. In contrast, constraint mapping focuses on translating specific queries other than the data values.

3) Discussion

We have proposed a generic type-based search-driven translation framework, which is well suited for the requirements of the on-the-fly constraint mapping among large scale data sources and our concern here is mainly focused on Bio-Medical Data.

V. Concluding discussion

This paper contains the core proposal we made through MetaQuerier architecture. Actually, the issue over here is that bio-medical data integration is an example of data integration. There are so many proposed data integration process. Our target is to deploy MetaQuerier as efficient data integration architecture and show one of its implementation. We proposed that we can use data integration for bio medical data or in the field of bio informatics. Here one part mainly gives idea about the MetaQuerier architecture and its sub processes and how each of the processes work. Although it's not focused how we can improve this MetaQuerier, we considered MetaQuerier is one most efficient data integration engine/design.

Inside the subsystem of metaquirer there are some conceptual changes of many common things to improve the efficiency of handling extremely huge and unsorted data. The three basic process of back-end were discussed elaborately. Other two processes are just briefly discussed. Our future work to make real time implementation based on some specific requirement and based on some ongoing bio medical research.

References

[1] Kevin Chen-Chuan Chang, Bin He, and Zhen Zhang. *Toward Large Scale Integration: Building a MetaQuerier over Databases on the Web.*

[2] Snehal Thakkar, Jos´e Luis Ambite, Craig A. Knoblock. *Composing, Optimizing, and Executing Plans for Bioinformatics Web Services.* In September 2, 2005.

[3] Gautam Pant, Padmini Srinivasan, and Filippo Menczer. *Crawling the Web.*

[4] Ping Wu, Ji-Rong Wen, Huan Liu, Wei-Ying Ma. *Query Selection Techniques for Efficient Crawling of Structured Web Sources.*

[5] Hai He, Weiyi Meng, Clement Yu, Zonghuan Wu. *WISE-Integrator: A System for Extracting and Integrating Complex Web Search Interfaces of the Deep Web*.

[6] Z. Zhang, B. He, and K. C.-C. Chang. On-the-fly constraint mapping across web query interfaces. In *Proceedings of the VLDB Workshop on Information Integration on the Web (VLDB-IIWeb'04)*, 2004.

[7] Bin He and Kevin Chen-Chuan Chang. *Automatic Complex Schema Matching Across Web Query Interfaces: A Correlation Mining Approach.*

[8] Chengyong Yang, Erliang Zeng, Tao Li, and Giri Narasimhan. *A Knowledge-Driven Method to Evaluate Multi-Source Clustering.*

[9] Hai He, Weiyi Meng, Clement Yu, Zonghuan Wu. *Automatic Extraction of Web Search Interfaces for Interface Schema Integration.*