Toward Large Scale Integration: Building a MetaQuerier over Databases on the Web

Kevin Chen-Chuan Chang, Bin He, and Zhen Zhang[*]

Computer Science Department
University of Illinois at Urbana-Champaign
{kcchang, binhe, zhang2}@uiuc.edu

Abstract

The Web has been rapidly "deepened" by myriad searchable databases online, where data are hidden behind query interfaces. Toward large scale integration over this "deep Web," we have been building the *MetaQuerier* system– for both exploring (to find) and integrating (to query) databases on the Web. As an interim report, *first*, this paper proposes our goal of the MetaQuerier for Web-scale integration– With its dynamic and ad-hoc nature, such large scale integration mandates both dynamic source discovery and on-the-fly query translation. *Second*, we present the system architecture and underlying technology of key subsystems in our ongoing implementation. *Third*, we discuss "lessons" learned to date, focusing on our efforts in system integration, for putting individual subsystems to function together. On one hand, we observe that, across subsystems, the system integration of an integration system is itself non-trivial– which presents both challenges and opportunities beyond subsystems in isolation. On the other hand, we also observe that, across subsystems, there emerge unified insights of "holistic integration"– which leverage large scale itself as a unique opportunity for information integration.

1 Introduction

In the recent years, the Web has been rapidly deepened with the prevalence of databases online. As Figure 1 conceptually illustrates, on this so-called "deep Web," numerous online databases provide dynamic query-based data access through their query interfaces, instead of static URL links. A July 2000 study [7] estimated 43,000-96,000 such search sites (and 550 billion content pages) on the Web. Our recent survey [11] (as we will explain) in April 2004 estimated

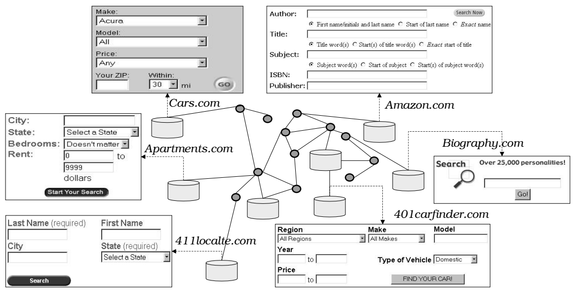


Figure 1: Databases on the Web.

450,000 online databases. As current crawlers cannot effectively query databases, such data are invisible to search engines, and thus remain largely hidden from users.

However, while there are myriad useful databases online, users often have difficulties in first *finding* the right sources and then *querying* over them. Consider user Amy, who is moving to a new town. To start with, different queries need different sources to answer: Where can she look for real estate listings? (*e.g.*, *realtor.com*.) Studying for a new car? (*cars.com*.) Looking for a job? (*monster.com*.) Further, different sources support different query capabilities: After source hunting, Amy must then learn the grueling details of querying each source.

To enable effective access to databases on the Web, since April 2002, we have been building a "metaquerying" system, the *MetaQuerier* (*metaquerier.cs.uiuc.edu*), as Figure 2 shows. Our goal is two fold– 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. Toward this goal, we have designed the system architecture, developed several key components, and started system integration. As an interim report, this paper presents our proposal of the MetaQuerier, summarizes its architecture and techniques, and reports lessons we have learned in putting things together.

Such deep-Web integration faces new challenges– for coping with the *large scale*: The deep Web is a large collection of queryable databases (well on the order 10^5, as mentioned earlier). As the large scale mandates, first, such integration is *dynamic*: Since sources are proliferating and evolving on the Web, they cannot be statically configured

[*]This material is based upon work partially supported by NSF Grants IIS-0133199 and IIS-0313260. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the funding agencies.

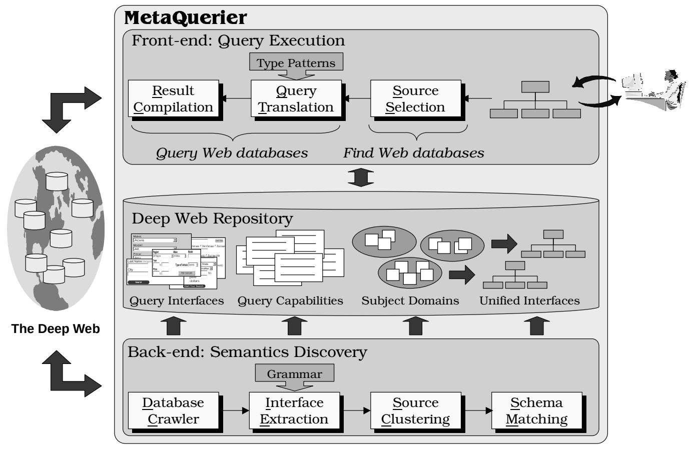


Figure 2: MetaQuerier: System architecture.

for integration. Second, it is *ad-hoc*: Since queries are submitted by users for different needs, they will each interact with different sources– *e.g.*, in Amy's case: those of real estates, automobiles, and jobs. Thus, toward the large-scale integration, the MetaQuerier must achieve dual requirements–

1. **Dynamic discovery**: As sources are changing, they must be dynamically discovered for integration– there are no *pre-selected* sources.
2. **On-the-fly integration**: As queries are ad-hoc, the MetaQuerier must mediate them on-the-fly for relevant sources, with no pre-configured *per-source* (*i.e.*, source-specific) knowledge.

We thus propose to build the MetaQuerier for dynamic discovery and on-the-fly integration over databases on the Web– To our knowledge, our goal of integration at a large scale has largely remained unexplored. While our research community has actively studied information integration (Section 2), it has mostly focused on static, preconfigured systems (say, Book comparison shopping over a set of bookstores), often of relatively small scale. In contrast, the MetaQuerier is motivated by the emergence and proliferation of Web databases for large-scale integration.

While the need is tantalizing– for effectively accessing the deep Web– the order is also tall. The challenge arises from the mandate of on-the-fly *semantics discovery*: Given the dynamically-discovered sources, to achieve on-the-fly integration, we must cope with various "semantics." To name a few: *What are the query capabilities of a source*? (So as to characterize a source and query it.) *How to match between query interfaces*? (So as to mediate queries.)

Such dynamic semantics discovery is crucial for our goal. While the challenge of semantics is not new to any information integration effort, for smaller and static scenarios, automatic semantics discovery is often an *option* to reduce human labor, as an aid to manually configured semantics (*e.g.*, source descriptions and translation rules). In contrast, for large scale scenarios, semantics discovery is simply a *mandate*, since sources are collected dynamically and queried on-the-fly. Is it, then, even hopeful to achieve such Web metaquerying?

Such integration is clearly hard– When we were starting, we were indeed also skeptical.[1] Nevertheless, to move forward, we decided to take two "strategies"– which have been beneficial in pointing us to the right direction.

As our *first* strategy, we were prepared to investigate not only the right techniques but also the *right goals* to achieve. We believe, sharing the insight of [8], that "as needs are so great, compromise is possible"– Given the scale of such integration, even "simpler scenarios," if can be achieved, will likely render great usage.

For our system goal, we thus decided to focus on a simpler scenario– integrating sources in the same domains. Note that each Web database provides structured information as a certain "type," or *domain*, of data– *e.g.*, *amazon.com* for "books" and *cars.com* for "automobiles." As the Web scales, many sources provide data in the same domains (e.g., there are numerous other "book" sources, such as *bn.com*). Such *domain-based* integration is simpler, as sources are more "homogeneous" in the same domain, and are thus more "integratable." On the other hand, such integration is indeed useful– Users mostly search for specific types of data. In our Amy's example, she needs access to sources in the Jobs domain (or Real Estates, Automobiles).

As our *second* strategy, we decided to get our *hands dirty* at the very beginning– to start with a "reality check" of the frontier, for guiding our research. We performed large scale surveys, by gathering random samples of Web servers over 1 million IP hosts, first in December 2002 [12] and then April 2004 [11]. The surveys clearly reassure the need for large scale integration: Our recent results [11] estimated 450,000 databases accessed by 1,258,000 query interfaces, across 307,000 deep-Web sites (thus a 3-7 times increase in 4 years compared to the 2000 study [7]).

For our development, these field studies have guided our research– To begin with, the surveys gathered a dataset of about 500 real sources, which serves as our test data, before we complete a "crawler" (Section 3) for dynamic source discovery. As a research project, we were thus able to take a "divide and conquer" approach, to identify and study key tasks concurrently. (As a "side effect," the dataset started our ongoing effort of constructing a shared data collection– the *UIUC Web Integration Repository*.[2])

Further, the surveys revealed some inspiring observations: Databases on the Web are *not* arbitrarily complex; there seem to be some "convergence" and "regularity" *naturally* emerging across many sources. This "concerted complexity" sheds light on the challenge of dynamic semantics discovery. (So, it is perhaps hopeful to achieve large scale metaquerying.) In hindsight, such behavior is indeed natural at a large scale: As sources proliferate, they tend to be influenced by peers– which we intuitively understand as the *Amazon effect*.[3] As Section 5 will discuss,

[1]Many colleagues in the community probably share this feeling– Our doubt actually started only after reading a long (5 pages) skeptical review, on the issue of finding "semantics," from an NSF panel of our original MetaQuerier proposal.

[2]Publicly available at *http://metaquerier.cs.uiuc.edu/repository*

[3]Online bookstores seem to follow *Amazon.com* as a de facto standard.

many of our approaches (Sections 3 and 4) have essentially built upon this very insight.

As an interim report, to begin with, this paper will present the system architecture and underlying technology of key subsystems in our ongoing implementation. To meet the dual requirements from dynamic discovery to on-the-fly integration, our system framework essentially "mines" all sources discovered to collect the semantics required for on-the-fly integration. Semantics discovery, as motivated earlier, is thus essential in the entire architecture.

Further, we discuss "lessons" learned to date, focusing on our efforts in *system integration*, for putting individual subsystems to function together.

Lesson 1: *Across subsystems, the system integration of an integration system is itself non-trivial*– which presents both challenges and opportunities beyond subsystems in isolation. To demonstrate the possibilities, we propose *ensemble cascading* and *domain feedback* as two sample techniques.

Lesson 2: *Across subsystems, there emerge unified insights of "holistic integration"*– which leverage the large scale itself as a unique opportunity for integration. This unified insight suggests holistic integration as a promising methodology for coping with the large scale.

We believe these "lessons" will not only guide our further development of the MetaQuerier, but also will be important principles for building large-scale integration systems in general. In the remainder of this paper, we start with Section 2 for the related work. We next, in Section 3, present the system architecture and underlying techniques. We then report lessons learned when "putting things together" in Sections 4 and 5. Section 6 concludes with open issues and future agenda.

2 Related Work

The MetaQuerier has a distinct focus as a large scale integration system– for dynamic discovery and on-the-fly mediation. On one hand, traditionally, information integration has assumed relatively small and pre-configured systems. On the other hand, the recent emergence of Web integration mostly considers only various subtasks. To our knowledge, this paper is the first one to present the overall system issues of building large scale integration.

Information integration has traditionally focused on relatively small-scaled pre-configured systems [15, 33] (*e.g.*, Information Manifold [25], TSIMMIS [31], Clio [30]). In particular, relevant issues on schema matching [14, 32, 28], schema mapping [35, 27], and query mediation [1, 2, 5, 6, 17] have been extensively studied.

In contrast, we are facing a "dynamic" and "ad-hoc" scenario (Section 1) of integrating databases on the Web. Such large-scale integration imposes different requirements and thus faces new challenges: To deal with this large scale, many tasks have to be automated, unlike integration at a small scale where sources can be manually "configured." First, for finding sources, we must dynamically select relevant sources according to user's ad-hoc information need, but not to fix a small set of sources for integration. Second, for modeling sources, we must automatically discover their query capabilities, but not to assume pre-configured wrappers providing source descriptions. Third, for querying sources, we must "on-the-fly" translate queries for unseen sources, but not to hard-code per-source knowledge specifically for each source. As Section 1 introduced, for realizing the MetaQuerier, these challenges essentially boil down to semantics discovery on-the-fly.

Further, more recently, many research efforts emerge to tackle various tasks for "Web integration"– *e.g.*, 1) query interface matching [18, 20, 34, 23], 2) query constraint mapping [10, 36], and 3) wrapper induction [4, 9, 13, 24]. While we share the interest in "component techniques," this paper studies the overall system architecture as well as issues and insights in system integration– Such "system" focus, we believe, is important in its own right.

In terms of our "solutions," we note that, as one of our unified insights (Section 5), the MetaQuerier exploits hidden "clues" revealed by holistic sources to discover underlying semantics. Along this line, several research efforts, which have also emerged recently, share this similar holistic insight– but specifically for the *schema matching* task, which we also address in the MetaQuerier [18, 20] (as Section 3 will report). In particular, references [34, 23] exploit clustering for holistically matching many schemas. Reference [26] proposes a "corpus-based" idea, which uses a separately-built schema corpus as a holistic "knowledge-base" for assisting matching of unseen sources.

While sharing similar holistic frameworks, in contrast to these efforts, we have developed our *holistic-integration* insight to generally tackle with "semantics discovery" common in many large-scale integration tasks, which we believe well generalize beyond the specific task of schema matching (*e.g.*, interface extraction and query translation), as Section 5 will discuss. Further, we believe, besides "statistical" analysis (which most other works have based upon), there are a wide range of applicable techniques (*e.g.*, syntactical parsing [37] for interface extraction; locality-based search [36] for query translation) to generally explore holistic hidden regularity for semantics discovery.

3 MetaQuerier: Architecture & Techniques

As Section 1 discussed, the integration of large scale databases on the Web calls for the needs of dynamic discovery and on-the-fly integration. Directed by these guidelines, we develop the MetaQuerier system architecture. Section 3.1 will overview the architecture, and Section 3.2 will briefly discuss the key techniques for the subsystems. Last, Section 3.3 will report our implementation status.

3.1 System Architecture

We design the MetaQuerier system as guided by the challenges of large scale integration. As Section 1 motivated, our scenario is essentially dynamic and ad-hoc: Sources are not pre-selected; there is no pre-configured per-source knowledge. Thus, the MetaQuerier must start from collecting databases on the Web (*i.e.*, dynamic discovery). However, while sources are dynamically discovered, at "run time," the MetaQuerier must query them (*i.e.*, on-the-fly integration) if selected. Our architecture design must essentially fill in the "semantics gap" from dynamically discovered sources to querying on-the-fly.

To begin with, we need to extract the query capability for each discovered query interface. Also, since, in our current development, we focus on integrating deep Web sources in the same domain, we need to develop a clustering approach to cluster interfaces into subject domains (*e.g.*, Books, Airfares). Last, for each domain, to construct a unified interface and translate the user's query from the unified interface to interfaces of specific sources, we need to discover semantic matchings among attributes. All these "minings" of semantics, while themselves necessary for large scale integration, in fact leverage the new opportunities of "holistic" integration, as Section 5 will elaborate.

Therefore, we design the MetaQuerier system as consisting of a *back-end* for semantics discovery and a *front-end* for query execution, which are connected by the *Deep Web Repository*, as Figure 2 illustrates. First, the back-end mines source semantics from collected sources. It automatically collects deep Web sources (*i.e.*, *Database Crawler* or subsystem *DC*), extracts query capabilities from interfaces (*i.e.*, *Interface Extraction* or *IE*), clusters interfaces into subject domains (*i.e.*, *Source Clustering* or *SC*) and discovers semantic matchings (*i.e.*, *Schema Matching* or *SM*). The collected query interfaces and discovered semantics form the Deep Web Repository, which will be exploited by the front-end to interact with users. (For our scenarios of dynamic and ad-hoc integration, we stress that such a Deep Web Repository is to be constructed "on-the-fly" after source discovery and modeling, without manually pre-configured source-specific knowledge.)

Second, in the front-end, our current design is to provide users a domain category hierarchy, which is similar to the category organization of *Yahoo.com*, but is automatically formed by the *SC* subsystem. For each category, a unified interface is generated using the *SM* subsystem. A user can thus first choose a domain of interest (*e.g.*, Books) and issue a query through the unified interface of that domain (*e.g.*, title *contain* `"database"` $\wedge$ subject = `"computer"`). The front-end then selects appropriate sources to query by matching their "capability" or "content" to the user query (*i.e.*, *Source Selection* or *SS*; Section 6 will discuss such query routing). With sources selected, the front-end further translates the user's query to interfaces of these sources (*i.e.*, *Query Translation* or *QT*) and aggregates query results to the user (*i.e.*, *Result Compilation* or *RC*).

Among the seven subsystems of the MetaQuerier (including both back-end and front-end), we have completed or partially completed five of them, *i.e.*, *DC* [11], *IE* [37], *SC* [21], *SM* [18, 20], and *QT* [36]. (The remaining *SS* and *RC* subsystems are in our future research agenda, as Section 6 will discuss.) For brevity, to highlight, in the following section, we will discuss three critical subsystems, *i.e.*, *DC*, *IE*, and *SM*, including their functionalities, key insights and approaches.

3.2 Subsystems: Key Techniques

Database Crawler [Subsystem *DC*]:

Functionality: As the first step of the MetaQuerier, the subsystem *DC* automatically discovers deep Web databases by crawling the Web and identifying query interfaces in Web pages. Query interfaces (in HTML format) will be passed to *IE* for constructing the repository of source query capabilities.

Insight: For discovering databases on the Web, completely crawling the entire Web is not only inefficient but also unnecessary– As online databases are accessed through query interfaces, we want to build a "focused" crawler that finds these "query interfaces" quickly. To guide our design of an effective "Web-database" crawler, we performed large scale surveys of the deep Web [11, 12] (as Section 1 mentioned). Our survey shows that query interfaces are often close to the root page of the Web site. That is, the *depth* of a Web database (*i.e.*, the minimum number of hops from the root page of the source site to a Web page containing a query interface of the Web database) is often very small. Specifically, by identifying and examining deep Web sites among 1,000,000 randomly generated IPs, we observed that no Web databases have depth more than 5 and 94% Web databases are within depth 3.

Approach: Motivated by this observation, we develop a *site-based crawler*, which focuses on examining shallow pages around the root page of a Web site. The site-based crawler consists of two stages: *site collector*, which finds valid root pages, and *shallow crawler*, which crawls pages within a Web server starting from a given root page. First, the site collector collects valid IPs that have Web servers: Since the IP space is huge and only a small fraction of 1/440 install Web servers [11], testing all the potential IPs is very inefficient. Therefore, we develop a site collector to quickly find IPs that host Web servers. This site collector is itself a crawler that traverses URLs by preferring out-of-site links, which thus gets to new site names quickly. Second, the shallow crawler crawls Web pages within a Web server (found by the site collector): Since query interfaces are close to the root page, the shallow crawler only needs to crawl a Web site up to a pre-configured depth (*e.g.*, 3).

Interface Extraction [Subsystem *IE*]:

Functionality: Given a query interface in its HTML format as collected by *DC*, the subsystem *IE* extracts the *query capability* in the interface. In particular, we view each query interface as consisting of a set of *constraint templates*, where a template specifies the "format" of an acceptable query condition, as a three-tuple [attribute; *operator*; value]. The task of *IE* is thus to extract the constraint templates of a given query interface. For example, the query interface QI_1 in Figure 4 should be extracted as four constraint templates, *i.e.*, S_1: [title; *contain*; \$v], S_2: [category; *contain*; \$v], S_3: [price range; *between*; \$low,\$high], and S_4: [reader age; *in*; {[4:8], ...}].

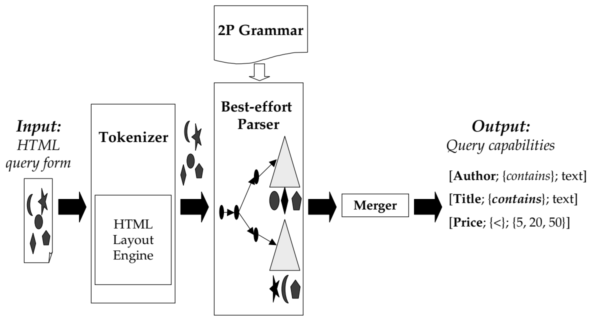


Figure 3: Subsystem *IE*: Interface extraction.

Insight: We observe that, query interfaces, although presented differently, often share similar or common query patterns. For instance, a frequently used pattern is a text (as the attribute name) followed by a selection list with numeric values (as the attribute domain), as the attributes reader age of QI_1, and age and price of QI_2 in Figure 4 show. Such observation motivates us to hypothesize the existence of a *hidden syntax* across holistic sources. That is, we rationalize the concerted structure by asserting the creation of query interfaces as guided by some hypothetical syntax: The hypothetical syntax guides a syntactic composition process from query conditions to their visual patterns. This hypothesis effectively transforms the problem into a new paradigm: We can view query interfaces as a *visual language* [29], whose composition conforms to a hidden, *i.e.*, *non-prescribed*, grammar. Their semantic extraction, as the reverse analysis, is thus a *parsing* problem.

Approach: We thus introduce a *parsing* paradigm by hypothesizing that there exists *hidden syntax* to describe the layout and semantic of query interfaces [37]. Specifically, we develop the subsystem *IE* as a visual language parser, as Figure 3 shows. 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. At its heart, we develop a *2P grammar* and a *best-effort parser*.

First, by examining many interfaces, a human expert summarizes and encodes two complementary types of presentation conventions as the 2P grammar. On one hand, we need to write *productions* to capture conventionally deployed hidden patterns. On the other hand, however, by capturing many patterns, some will conflict, and thus we also need to capture their conventional precedence (or "priorities") as *preferences*.

Second, to work with a hypothetical syntax, we develop our parser to perform "best-effort." As a non-prescribed grammar is inherently ambiguous and incomplete, we need a "soft parsing" semantics– The parser will assemble parse trees that may be multiple (because of ambiguities) and partial (because of incompleteness), instead of insisting on a single perfect parse. On one hand, it will prune ambiguities, as much as possible, by employing preferences (as in the 2P grammar). On the other hand, it will recognize the structure (by applying productions) of the input form, as much as possible, by maximizing partial results.

When there are multiple parse trees for the same query interface, we need an error handling mechanism to generate the final output. While the parser framework is rather generic, error handling is often application specific. As our "base" implementation, our "Merger" (Figure 3) simply merges all query conditions covered in all parse trees, to enhance the "recall" (or coverage) of extraction. In our subsequent system integration of putting subsystems together, we observed further opportunities for error resolution, as Section 4 will discuss.

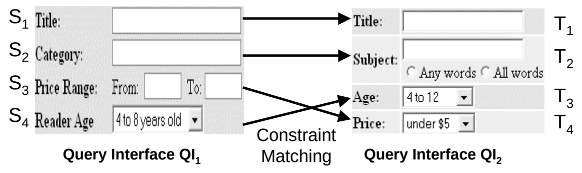


Figure 4: Example query interfaces and their matching.

Schema Matching [Subsystem *SM*]:

Functionality: For each domain, the subsystem *SM* discovers semantic correspondences (*i.e.*, matchings) among attributes in the extracted query capabilities. For instance, in Books domain, we may find subject is the synonym of category, *i.e.*, subject = category. In particular, we generally consider to discover complex matchings. In contrast to simple 1:1 matching, complex matching matches a set of m attributes to another set of n attributes, which is thus also called $m{:}n$ *matching*. For instance, in Books domain, author = {first name, last name}; in Airfares domain, passengers = {adults, seniors, children, infants}.

The discovered matchings are stored in the Deep Web Repository and serve for two purposes: 1) They are exploited to construct a unified interface for each domain, which is presented to users at the front-end. 2) They are used to match attributes from the unified interface to the selected sources (by *SS*)– The subsystem *QT* needs such matchings as input to translate the user's query.

Insight: Existing schema matching works mostly focus on small scale integration by finding attribute correspondences between two schemas and thus are not suitable for matching among many sources [32, 28, 14]. To tackle the challenge of the large scale matching, as well as to take advantage of its new opportunity, we propose a new approach, *holistic schema matching*, to match many schemas at the same time and find all the matchings at once. Such a holistic view enables us to explore the *context* information across all schemas, which is not available when they are matched only in pairs.

In particular, we started by exploring attribute occurrences across sources as the context and proposed the MGS matching approach with the assumption of the existence of a hidden generative schema model, which generates query interfaces from a finite vocabulary of attributes [18]. In our further study, and in our current implementation, we explore the co-occurrence patterns of attributes to discover complex matchings [20]. For instance, we may observe that last name and first name have a high probability to co-occur in schemas, while they together rarely co-occur with author. More generally, we observe that *grouping* *attributes* (*i.e.*, attributes in one group of a matching *e.g.*, {last name, first name}) tend to be co-present and thus positively correlated across sources. In contrast, *synonym attributes* (*i.e.*, attribute groups in a matching) are negatively correlated because they rarely co-occur in schemas.

Approach: This insight motivates us to abstract the schema matching problem as correlation mining [20]. Specifically, we develop the DCM approach for mining complex matchings, consisting of automatic data preparation and correlation mining. As preprocessing, the data preparation step cleans the extracted query capabilities to prepare "schema transactions" for mining. Then the correlation mining step discovers complex matchings with dual correlation mining of positive and negative correlations.

3.3 Implementation Status

For building the MetaQuerier (Figure 2), as a research project, we aim at delivering along the way. To begin with, we take a *divide-and-conquer* approach: As Section 1 motivated, we develop the key components concurrently– To date, we have studied and implemented *DC* [11], *IE* [37], *SC* [21], *SM* [18, 20], and *QT* [36]. To speed up prototyping, first, we have decided to leverage open-source software whenever possible. For instance, the site-based crawler *DC* embeds the open source wget [16] as its base crawler, and adds on its new crawling logic. Also, *IE* needs to use an "HTML layout engine" (Figure 3) for rendering visual tokens– We built this tokenizer upon the DOM API, which many Web browsers (*e.g.*, Mosaic or Internet Explorer) support. Second, we decided to mainly use scripting languages. In particular, we use Python whenever possible– However, the mix of open source code also brought in hybrid languages, *e.g.*, C for wget.

Further, we aim at incremental deployment, to "package" smaller scopes in the course of development, before the complete MetaQuerier. To date, we have deployed two "applications." First, for *Web crawling*: We used (part of) *DC* in our deep Web surveys [11, 12], as Section 1 mentioned. Second, for *query-interface integration*: We have assembled two critical subsystems, *IE* and *SM*, for automatically extracting and unifying Web query interfaces, which we demonstrated in [22]. In such system assembly, when putting things together, we observed several "lessons," which we next report in Section 4 and 5.

4 Putting Together: Integrating Subsystems

Toward building the MetaQuerier system, we are intrigued to learn that the "system integration" of such an integration system is itself non-trivial– Beyond each individually validated subsystem, putting things together actually brings forward new issues– with not only challenges but also opportunities. This section reports the issues we have observed and sketches our solutions.

As just discussed, for our system effort, we naturally took a "divide-and-conquer" approach; we identify the key tasks, and develop each corresponding subsystem in its

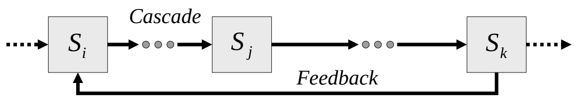


Figure 5: System integration: Putting subsystems together

abstraction, *e.g.*, interface extraction [37], schema matching [18, 20], source clustering [21], and query translation [36]. As we started with and focused more on subsystems, we expected that putting things together is probably no more than straightforward interfacing and assembly of modules.

However, such tasks for information integration are often of a "soft" nature– Rather than executing well-defined semantics, their effectiveness is measured by *accuracy*. Such accuracy is usually subjective and context-dependent. First, *how accurate is good enough?* Is it good enough if a single subsystem S_i delivers "90%" accuracy? The answer depends on the "consumer" of S_i's output (say, S_k). Second, *can we make it even more accurate?* While techniques matter, the accuracy also depends on the availability of more clues (*e.g.*, from other subsystems) to leverage.

We thus observed that, when putting subsystems together, this "system integration" presents new issues. As Figure 5 conceptually illustrates, in our experience to date, we found both opportunities and challenges: On one hand, the *challenge* of *cascade*, assembling subsystems may demand higher accuracy: it may turn out that the accuracy of S_i does not sustain a subsequent S_j. On the other hand, the *opportunity* of *feedback*, putting subsystems together may supply more information: it may turn out that the information available at a latter stage, say S_k, can help to enhance the accuracy of S_i.

Specifically, we will discuss the materialization of such issues, centering around the interface extraction subsystem *IE*. When studied in isolation (Section 3.1), *IE* delivers 85-90+% accuracy (as our earlier work [37] reported)– thus it will make about 1-1.5 mistake for every 10 query conditions to extract. While seemingly satisfactory, putting in the context of the entire system (Figure 2), is this accuracy good enough? Can it be made more accurate? Beyond the isolated study, in our system integration, we realized that 1) the accuracy does not sustain the schema matching task *SM* in cascade, and 2) it can indeed benefit from feedback for further error handling. More specifically, taking *IE* as an example, we will propose two "sample" system-integration techniques– focusing on their intuitive insights. Our proposal is only a staring point– As Section 6 will discuss, we believe such system integration is important for integrating systems in its own right, and we plan to study more systematically and thoroughly.

- **Ensemble cascading**: On one hand, to sustain the accuracy of *SM* under imperfect input from *IE*, we develop the *ensemble* framework, which aggregates a multitude of executions of "base executions" to achieve robustness. (Section 4.1).

- **Domain feedback**: On the other hand, to take advantage of information in latter subsystems, we develop the *feedback* framework, which improves the accuracy of *IE* by exploiting the domain statistics acquired from schema matching (Section 4.2).

Before discussing these two techniques, we first present and analyze the origin of the problem: the extraction errors in the output of *IE*.

Preliminary: Error Handling of Interface Extraction

While the parsing approach to extracting interfaces can achieve over 85% accuracy, the 15% errors may still significantly affect the matching quality. Figure 6(a) shows the *base* framework of integrating the *IE* and *SM* subsystems by simply concatenating them. As our experiment reports [19], with such a base framework, the errors in the interface extraction may affect matching accuracy up to 30%.

Specifically, as the 2P grammar is inherently ambiguous, it is possible to have multiple parse trees for the same query interface. Different interpretations of the same token lead to the conflicts of different parses. An incorrect parse may associate tokens in a wrong way. Figure 8 illustrates several such cases, where each circled group represents an association of tokens as a query condition. For instance, Figure 8(b) associates a selection list with either "Last Name" or "e.g., Mike," and thus these two associations conflict on the selection list. Without semantic annotation, both associations are valid patterns according to the 2P grammar.

As Section 3.2 explained, as our base implementation, when *IE* is independently developed, we handle the errors by simply taking a union of all the parse trees as the final output. However, when integrated into the system context, the errors in *IE* may affect the accuracy of subsequent subsystems. On the other hand, the conflicts in *IE* can be resolved by considering the feedback from other subsystems.

4.1 Cascading: The Ensemble Framework

As our first attempt for integrating the MetaQuerier, we cascade two important subsystems *IE* and *SM*, with the input of *IE* as a set of manually collected query interfaces in the same domain. Also, as discussed in Section 3.2, our base algorithm for *SM* is essentially a holistic matching framework that "mines" semantic correspondences among attributes as positive and negative correlations. *SM* thus takes a set of schemas as input and outputs a ranked list of matchings evaluated by some correlation measure.

We notice that the performance degradation with the base framework (Figure 6(a)) results mainly from the negative impact of the noisy input on the right ranking of matchings in the output of *SM*. When input schemas are noisy, the ranking of matchings is likely to be affected (*i.e.*, incorrect matchings maybe ranked higher than correct ones). Consequently, the ranking is less reliable for the "consumer" applications of *SM* to select correct matchings. For instance, an application-specific matching selection step is often introduced after *SM* to choose the most promising subset of matchings among all the discovered ones. Since such a selection naturally relies on the ranking of matchings, it is critical to make *SM* still output a good ranking with the presence of noises.

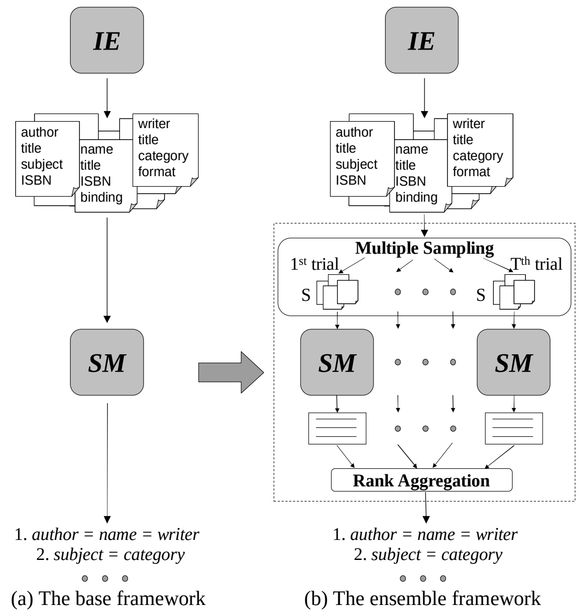


Figure 6: Integration of subsystems *IE* and *SM*.

While large scale data integration brings forward the inherent problem of noisy quality in interface extraction, the large scale also lends itself to an intriguing potential solution. An interesting question to ask is: *Do we need all input schemas in matching their attributes*? In principle, since pursuing a data mining approach, the holistic matcher exploits "statistics-based" evaluation (*e.g.*, correlation mining in our case) in nature and thus needs only "sufficient observations." As query interfaces tend to share attributes, *e.g.*, author, title, subject, ISBN are repeatedly used in many book sources, a subset of schemas may still contain sufficient information to "represent" the complete set of schemas. Thus, the holistic matcher in fact only needs sufficient correct schemas to execute, instead of all of them. This insight is promising, but it also brings a new challenge: As there is no way to differentiate noisy schemas with correct ones, how should we select the schemas to guarantee the robustness of our solution?

Tackling this challenge, we propose an *ensemble* framework, with sampling and voting techniques, to build upon and extend our holistic matcher, and meanwhile maintain its robustness. To begin with, we consider execute the holistic matcher on a randomly sampled subset of input schemas. Such a *downsampling* has two attractive characteristics: First, when schemas are abundant, it is likely to contain sufficient correct schemas to be matched. Second, by sampling away some schemas, it is likely to contain fewer noises and thus has more chances to sustain the holistic matcher. We name a downsampling as a *trial*.

Further, while a single trial may (or may not) achieve good result, as a randomized scheme, the expected robustness can only be realized in "statistical" sense– Thus, we propose to take an ensemble of multiple matchers, where each matcher is executed over an independent trial of schemas. We expect the majority of these matchers have better results than directly run the matcher on all the schemas. Thus, by taking majority voting among these matchers, we can achieve a much better matching accuracy.

Therefore, the ensemble framework consists of two steps: *multiple sampling* and *rank aggregation*, as Figure 6 illustrates. The multiple sampling step randomizes the input schemas with multiple T trials, where each trial is a downsampling of the input schemas with sampling size S. We then execute the DCM holistic matcher on each trial. The output of each trial is a list of ranked matchings. Finally, the rank aggregation step aggregates the discovered matchings from all the trials into a merged list of ranked matchings by taking majority voting.

For brevity of exposition, we intuitively motivate the effectiveness of this ensemble framework.

Example 1: Suppose there are 50 input schemas. Suppose a matching M cannot be correctly ranked because there are five noisy schemas that can affect M. On the other hand, assume M can be correctly ranked if there are no more than two noisy schemas. Also, suppose we want to sample 20 schemas in one trial and conduct 100 trials in total (*i.e.*, S = 20 and T = 100).

By simple derivation, we can see that we have 0.07 probability to get a single trial with no noisy schemas, 0.26 probability with one and 0.36 probability with two. Together, we have 0.07 + 0.26 + 0.36 = 0.69 probability to correctly rank M in a single trial (*i.e.*, when there is no more than 2 noises), denoted by $Pr(M) = 0.69$.

Next, we are interested in how many times, among the 100 trials, can we observe M being ranked correctly? This problem can be transformed as a standard scenario of tossing an unfair coin in statistics: Given the probability of getting a "head" in each toss as $Pr(M)$, with 100 tosses, how many times can we observe heads? With this equivalent view, we know that the number of trials in which M is correctly ranked (*i.e.*, the number of tosses to observe heads), denoted by $O(M)$, is a random variable that has a binomial distribution [3] with the success probability in one trial as $Pr(M)$. Figure 7 shows the binomial distribution of $O(M)$. This figure essentially shows, if the probability to get a head in one toss is 0.69, after tossing 100 times, the probability of observing a certain number of heads.

As we take majority voting among all the trials, we are thus interested in the probability that we can correctly rank M (or observe heads) in the "majority" (*i.e.*, more than 50) of any 100 trials (or tosses). From Figure 7, we know that we have 0.9996 probability to observe M in no less than 50 trials, which means it is almost for sure that M can be correctly ranked in the majority of trials. ■

Example 1 analytically illustrates, when S and T are determined, the effectiveness of the ensemble framework. However, to realize this framework, we still need to tackle some challenges in each step. First, in the multiple sampling steps, we need to develop a principled approach to automatically determining an appropriate setting of the sampling size and the number of trials that can guarantee good robustness of a holistic matcher. Second, in the rank aggregation step, we need to develop an aggregation strategy

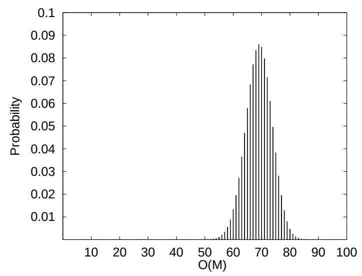


Figure 7: The binomial distribution of $O(M)$, with 100 trials and $Pr(M) = 0.69$.

that can reflect the wills of the majority in the merged list of matchings– That is, if a matching can be correctly ranked in most trials, its ranking in the merged result should also be correct. Please refer to [19] for these technical details.

To evaluate the matching performance, we compare the accuracies of the base framework and the ensemble framework on matching query interfaces in two domains, Books and Airfares, with DCM as the matching algorithm. The experiments show that the ensemble framework can improve the matching accuracy of DCM by up to 20%.

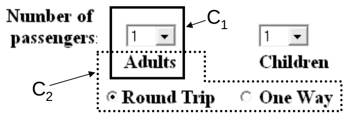
4.2 Feedback: Domain Statistics

While the *ensemble* just discussed addresses how *SM* can robustly cascade with *IE*, this section explores the possibility where *IE* may take advantage of the *feedback* of more "clues," in this case domain statistics, from *SM* to correct the errors of *IE*.

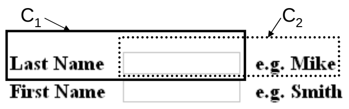
Example 2: Figure 8(a) highlights a conflict raised during parsing of a query interface, where two constraints: C_1 = [adults; *equal*; \$val:{1,2,...}] and C_2 =[adults; *equal*; \$val:{round-trip, one-way}] compete for the attribute adults. After examining a collection of interfaces in the same domain, *SM* identifies that constraint adults is most likely of `numeric` type, which has very distinctive patterns such as a numeric selection list shown in Figure 8(a). Such a notion of *type* with *distinctive patterns* can often be observed in query interfaces, *e.g.*, numeric (as just observed) and date (with day, month, year inputs). Such type information, if available, is very helpful for conflict resolution. In this particular example, knowing the attribute adults is more likely of numeric type, we are able to determine more confidently that C_1 is a better association. ■

In general, *IE* can leverage the collective statistics of the peer query interfaces in the same domain. While *IE* handles one interface at a time (Section 3.2), such "holistic" domain statistics is often available at subsequent subsystems, where many interfaces gather for further processing. Specifically, as *SM* takes interfaces from the same domain as input, it is able to collect such statistics in the process of discovering matchings.

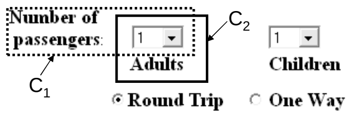
Note that, as Example 2 hints, our exploration of "domain statistics" is itself a "voting" (or majority) strategy– The conflicting associations are resolved by all "peer" query interfaces in the same domain. For this example, by voting on possible attribute types, we can conclude that



(a) Conflict 1 in query Interface QI_a



(b) Conflict 2 in query Interface QI_b



(c) Conflict 3 in query Interface QI_a

Figure 8: Conflict examples

adults is more likely to occur in a numeric pattern and thus choose the correct association. Section 5 will further "unify" the domain-feedback scheme with ensemble-cascading (Section 4.1) by the same insight of "majority."

In our development, we identify three types of domain statistics as feedback information that are effective in resolving the conflicts remained from parsing.

Type of attributes

As illustrated in Example 2, there is a notion of type underlying many attributes, which often reveals distinctive patterns. Since matched attributes are usually of the same type, by looking at many interfaces, *SM* is able to collect the common type of an attribute, as one type of domain statistics.

Frequency of attributes

Each domain typically has a set of frequently queried attributes. For example, author, title, ISBN, subject, category are the common attributes in Books domain, and departure city, departure date, passengers, adults, children in Airfares domain. A reasonable assumption we can make is that a query interface is more likely to ask queries on those common attributes. Therefore, the information of attribute frequencies, as another type of domain statistics, can be used to resolve conflicts.

Example 3: Consider a conflict generated in interpretation of query interface QI_b as shown in Figure 8(b). Constraint C_1 =[last name; *contain*; $val] and C_2 =[e.g.Mike; *contain*; $val] conflict on the input box represented as $val. However, as *SM* collects constraints from many query interfaces, it may tell *IE* that attribute last name is much more frequently queried by the peer interfaces, while attribute e.g.Mike rarely occurs. We thus may conclude that constraint C_1 is preferred than C_2. ■

Correlation of attributes

As discussed in Section 3.2, *SM* discovers matched attribute groups, where attributes within the group are positively correlated, and attributes across the groups negatively correlated. As the third type of domain statistics, such correlations can serve as the constraints to check the sanity of the extracted results and further to resolve conflicts.

Example 4: Consider the conflict between number of passengers C_1 and adults C_2 in QI_a in Figure 8(c). *SM* may discover that attributes adults and children are positively correlated, while both of them are negatively correlated with passengers. Given children being confidently identified as an attribute (*i.e.*, with no conflicts with others), choosing adults will be consistent with the discovered correlations, while favoring passengers against it. Therefore, we may consider adults as a better interpretation than passengers. ■

Given these three types of feedback information, we need to combine them with a coherent mechanism for enhancing the accuracy (as further error handling beyond what Section 3.2 discussed) of *IE*. Currently, we explore a simple rule based framework. Specifically, we write the conflict resolution policies as rules, where each rule states the condition to be satisfied, and the favored constraint to be chosen by exploiting the domain statistics. In particular, we have three preference rules, with each rule reflecting one type of domain statistics:

- *Rule 1: Using type of attribute*. This rule favors the constraint whose type is more frequently used in other interfaces.
- *Rule 2: Using frequency of attribute*. This rule favors the constraint whose attribute is more frequently queried.
- *Rule 3: Using correlation of attribute*. The rule favors the constraints that co-occur with any positively correlated constraint, and disfavors the constraints that co-occur with any negatively correlated constraint.

While those preferences, when working individually, determine a favored constraint, their choices may not agree with each other. For instance, when resolving the conflict between adults and passengers in Figure 8(c), if we deploy rule 3, we will favor adults. However, suppose that passengers is more frequently queried than adults, deploying rule 2 will give a different answer, which favors passengers. To solve this problem of inconsistency, we adapt a simple strategy – we prioritize the three rules as rule 3→2→1, based on the confidence of the correctness of each rule.

To validate the effectiveness of using the feedback information, we conducted a preliminary study to see how much the domain statistics can help to improve the accuracy of IE. In particular, we collected the parsed results in Airfares and Books domains, with 20 and 30 interfaces respectively. We measured the results as the percentage of conflicts that are resolved correctly using the feedback information. The result from the Airfares domain shows among the 20 airline query interfaces, there are 7 conflicts and all of them can be correctly resolved. For Books domain, there are 7 conflicts among 30 book query interfaces and 4 out of the 7 are resolved. Totally, 11 out of 14 conflicts from the 50 interfaces are correctly resolved, which amounts to 78.6% percentage. This preliminary study shows that exploring feedback information has good potentials for improving accuracy.

Although the rule based approach is rather simple, it tends to be heuristic. A more principled way, which we are currently investigating is to explore a probabilistic model that combines all three types of domain statistics together to generate an overall best, as the "most-probable," interpretation. Such a probabilistic model is motivated by our observation that the feedback information essentially denotes the likelihood of a constraint to be correct from different aspects (*i.e.*, type, frequency and correlation), which can be estimated by the domain statistics we have collected. Under this view, resolving conflicts thus becomes choosing the interpretation with highest probability.

5 Putting Together: Unified Insights

Toward building the MetaQuerier system, we are also inspired to observe that there seem to emerge common insights across subsystems. While we have developed these tasks (Section 3.2) separately, each with its specific techniques, putting things together actually reveals a common methodology, which conceptually unifies the seemingly different approaches. This section discusses this methodology of *holistic integration* and the insights it implies.

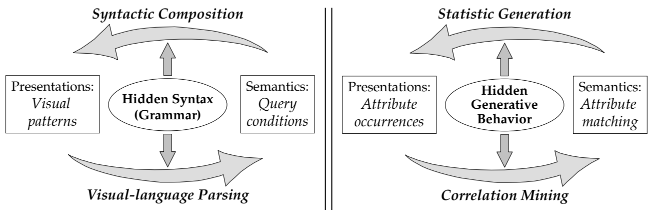
To begin with, as Section 1 motivated, we note that any integration task is, to a large extent, about *semantics discovery*– to discover certain target *semantics*: *e.g.*, for task *IE* (interface extraction): "understanding" query conditions; for *SM* (schema matching): "matching" them. The major barrier for large scale integration, with its dynamic and on-the-fly nature, is exactly such semantics discovery, for the lack of pre-configured per-source knowledge.

By "holistic integration," we take a holistic view to account for *many* sources together in integration, by globally exploiting clues across all sources for resolving the "semantics" of interest– To our surprise, although not obvious by their own, when put together, many of our integration tasks implicitly share the same holistic-integration framework– which thus conceptually "unifies" our various techniques. As a hindsight, we thus "propose" holistic integration as a conceptual framework and a unified methodology for large scale integration.

Specifically, we observed two common insights in the various materializations of holistic integration across the MetaQuerier system, in both the techniques for subsystems (Section 3) and system integration (Section 4).

- **Hidden regularity**: Holistic integration can leverage *hidden regularity* across many sources, to discover the desired semantics– For our subsystems (Section 3), tasks *IE* exploits hidden "syntax" and *SM* hidden "schema model," as Section 5.1 will explain. (While not explained here, *SC* and *QT* also explore the same concept.)
- **Peer majority**: It can also leverage *peer majority*, by taking clues from the majority of peers– For our system integration (Section 4), the ensemble-cascading and domain-feedback schemes both exploit the majority for error handling, as Section 5.2 will explain.

As evident from our experience (albeit limited), we believe that holistic-integration is promising for large scale integration, by leveraging the challenge of scale as an opportunity: We are inspired that large scale can itself be a


Syntactic Composition
Presentations: Visual patterns
Hidden Syntax (Grammar)
Semantics: Query conditions
Visual-language Parsing
Statistic Generation
Presentations: Attribute occurrences
Hidden Generative Behavior
Semantics: Attribute matching
Correlation Mining

(a) *IE*: interface extraction **(b)** *SM*: schema matching

Figure 9: Holistic integration: Exploring regularity

crucial leverage to solve integration tasks– By exploring the hidden regularity and peer majority across many sources, a holistic approach will take advantage of the large scale (with sufficient "samples") to discover the desired semantics for integration. These "holistic" insights, we believe, will be essential for large scale integration tasks.

5.1 Hidden Regularity: Semantics Discovery

As the first leverage, by holistic integration, we explore hidden regularity existing across sources. As just discussed, any integration task is essentially the discovery of certain target semantics– but, we can only observe some "surface" *presentations*. As a unified insight, several of our subsystems– we will use *IE*, *SM* as examples– have exploited the hidden regularity of surface presentations for semantics discovery. In retrospect, we observe that, under the same holistic-integration spirit, our subsystems have built upon two common hypotheses, which relate underlying semantics to observable presentations, across many sources.

($\mathcal{S}$) *Shallow observable clues:* The "underlying" semantics often relates to the "observable" presentations, or shallow clues, in some way of *connection*. Thus, we can often identify certain observable clues, which reflect the underlying semantics.

($\mathcal{H}$) *Holistic hidden regularity:* Such connections often follow some implicit properties, which will reveal holistically across sources. Thus, by observing many sources, we can often identify certain hidden regularity that guides how the semantics connects to the presentations.

These hypotheses shed light for dynamic semantics discovery (Section 1) in large scale integration: By identifying the holistic regularity, our integration task, to discover the desired semantics, is thus the *inverse* of this semantics-to-presentations connection. That is, our "holistic integration" framework can tackle large scale integration by developing some *reverse analysis*, which holistically analyzes the shallow clues, as guided by the hidden regularity, to discover the desired semantics. This general framework conceptually unifies our approaches for several tasks– We now demonstrate with *IE* and *SM*, as Figure 9 contrasts.

First, consider task *IE*: As Section 3.2 introduced, the observation of condition "patterns" motivated us to hypothesize the existence of *hidden syntax*– which, in our term now, is the hidden regularity, across holistic sources. As Figure 9(a) shows, the hypothetical syntax (as *hidden regularity*) guides a syntactic composition process (as *connection*) from query conditions (as *semantics*) to their visual patterns (as *presentations*). That is, there exists a syntactic connection (Hypothesis $\mathcal{S}$), and such connections at various sources share the same grammar as the regularity (Hypothesis $\mathcal{H}$). This hidden syntax effectively transforms the problem: As Section 3.2 described, we view query interfaces as a *visual language*; their extraction is precisely the reverse analysis– or *visual-language parsing*.

Second, consider task *SM*. As Section 3.2 introduced, we hypothesize a hidden generative behavior, which probabilistically generates, from a finite vocabulary, the schemas we observed– In our term now, this consistent generative behavior is the hidden regularity. As Figure 9(b) shows, the hidden generative behavior (as *hidden regularity*) guides a statistic generation process (as *connection*) from attribute matching (as *semantics*) to their occurrences in interfaces (as *presentations*). That is, there exists a statistic connection (Hypothesis $\mathcal{S}$), and such connections at various sources share the same generative behavior as the regularity (Hypothesis $\mathcal{H}$). This generative behavior constrains how attributes may occur in interfaces–*e.g.*, grouping attributes tend to positively co-occur while synonym attributes negatively. The *reverse analysis* to find attribute matchings is thus the "mining" of correlated attributes, and thus a *correlation mining* approach.

5.2 Peer Majority: Error Correction

As the second leverage, by holistic integration, we explore the *majority* of peers for correcting errors made by the relatively *few*. As Section 4 noted, any integration task, in its semantics discovery, is essentially "soft," since it can make errors– Thus, we generally need *error correction*, for correcting errors of a "base algorithm," to enhance its accuracy. In the MetaQuerier, such error correction arises in our system integration (Section 4).

This section will propose, as we observed, that such error correction can also leverage the holistic view across many sources. The insight hinges on the following hypotheses, which concern executing some base algorithm over a data "instance" as input.

($\mathcal{B}$) *Reasonable base:* The base algorithm is "reasonable"– While not perfect, errors are relatively rare.

($\mathcal{R}$) *Random samples:* The data instance can be randomly sampled, for the base algorithm to execute over.

These hypotheses hint on a majority-based approach for error correction in large scale integration: Let's create many samples of base results. First, each base sample will make rare errors, since the base is "reasonable" (Hypothesis $\mathcal{B}$). Second, the errors across samples are independent, since the data instances are "random" (Hypothesis $\mathcal{R}$). Together, by counting over all samples, we can use the majority among them to correct the relatively few errors. (Such "boosting" can be formally derived, with the same intuition as Example 1.) This general approach conceptually unifies our error-correction schemes (of Section 4).

First, consider ensemble-cascading. As Section 4.1 proposed, it enhances the accuracy of *SM* (as the *base algorithm*) for matching input query schemas (as the *data instance*). As Figure 6 shows, the ensemble scheme creates multiple samples (Hypothesis $\mathcal{R}$) of the base results, by downsampling the original input. By design (Section 3.2), the base *SM* is "reasonable" in finding correct matchings (Hypothesis $\mathcal{B}$). We thus take a majority voting, which enhances the accuracy of *SM*.

Second, consider domain-feedback. As Section 4.2 proposed, it enhances the accuracy of *IE* (as the *base algorithm*) for extracting query conditions (as *data instance*) in an interface. In our holistic framework, we run *IE* for *all* interfaces the crawler discovered (Figure 2)– Thus, we naturally create multiple "samples" for any query condition; *e.g.*, a condition on adults will likely appear in many interfaces, in different ways– each is thus a random sample (Hypothesis $\mathcal{R}$). By design (Section 3.2), the base *IE* is "reasonable" and thus extracts correctly most of the time (Hypothesis $\mathcal{B}$). The feedback mechanism will gather statistics from all samples, for correcting errors– Such statistics (*e.g.*, the likely type of attribute adults in the Airfares domain) reflects the majority– similar to voting.

6 Concluding Discussion: Issues & Agenda

As an interim report, this paper presents our proposal of the MetaQuerier, summarizes its architecture and techniques, and reports lessons we have learned. While we are still to deliver our promise of a complete MetaQuerier, our experience has been encouraging– As this paper has presented, our course of development has gained valuable insights for large scale integration, which will continue to direct us toward building the MetaQuerier– As our future agenda, there remain many open issues, as we discuss next.

To begin with, while we have developed subsystems *DC*, *IE*, *SC*, *SM* and *QT*, to complete the entire system (Figure 2), we need to further develop the currently missing components. *First*, to route a user's query to right sources, the subsystem *SS* needs to support an effective and efficient source selection strategy. Such selection will likely call for more sophisticated *source modeling*, to capture not only query capability (as we currently do) but also data quality of a Web database. With such modeling, we will develop scoring schemes for ranking sources by their potential to "satisfy" a query. *Second*, to present query results, the subsystem *RC* needs to compile the results from different sources into a coherent piece. Such compilation will require extracting data from the result pages and matching objects across different sources, among other issues. While existing works on wrapper induction (Section 2) have extensively studied such extraction, our scenarios again allow us to leverage the "holistic" insight, *e.g.*, in a way similar to how *IE* builds upon "hidden syntax." We plan to complete our study of all the modules.

Further, beyond individual subsystems, we believe the "science" of system integration, for building an integration system like the MetaQuerier, deserves thorough study. As Section 4 suggested, such integration is not simply module assembly– There are interesting architectural issues to study. While we have proposed ensemble-cascading and domain-feedback as sample techniques, the "science" is not yet clear. For example, will cascading and feedback coexist? In that case, will there be some stable "fixpoint" in the "feedback loop"– Note that Figure 5 clearly resembles similar loop structure in *control theory*. We wonder how much we can borrow from the discipline to design a "feedback" integration system. We plan to more systematically study this "science" of system integration.

Finally, as we move closer to system completion, we will validate with large scale crawling of Web databases. As Section 1 explained, we currently rely on our test data repository for concurrently developing various tasks– While our subsystems seem to perform well in their isolated study (Section 3.2) on our test dataset, will they indeed scale to the real Web scale (for sources on the order of 10^5)? We have started crawling the deep Web, which will significantly push our "scale" of study.

As we conclude, we are eager to further our exploration of large scale integration over the deep Web– As we move forward, while unforeseen challenges will likely arise, we are optimistic that inspiring insights will again emerge.

References

[1] S. Abiteboul and O. M. Duschka. Complexity of answering queries using materialized views. In *ACM PODS*, pages 254–263, 1998.

[2] F. N. Afrati, C. Li, and J. D. Ullman. Generating efficient plans for queries using views. In *ACM SIGMOD 2001*, pages 319–330, 2001.

[3] D. R. Anderson, D. J. Sweeney, and T. A. Williams. *Statistics for Business and Economics (Second Edition)*. West Pub. Co., 1984.

[4] A. Arasu and H. Garcia-Molina. Extracting structured data from web pages. In *SIGMOD Conference*, 2003.

[5] Y. Arens, C. A. Knoblock, and W.-M. Shen. Query reformulation for dynamic information integration. *J. Intell. Inf. Syst.*, 6(2-3):99–130, 1996.

[6] C. Beeri, A. Y. Levy, and M.-C. Rousset. Rewriting queries using views in description logics. In *Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 99–108, 1997.

[7] M. K. Bergman. The deep web: Surfacing hidden value. Technical report, BrightPlanet LLC, Dec. 2000.

[8] M. Carey and L. Haas. The top 10 reasons why federated can't succeed– and why it will anyway. *Presentation*. The Lowell Database Research Self-Assessment Meeting, May 2003. Available at `http://research.microsoft.com/∼gray/lowell`.

[9] J. Caverlee, L. Liu, and D. Buttler. Probe, cluster, and discover: Focused extraction of qa-pagelets from the deep web. In *ICDE Conference*, 2004.

[10] K. C.-C. Chang and H. García-Molina. Conjunctive constraint mapping for data translation. In *Proceedings of the 3rd ACM International Conference on Digital Libraries*, 1998.

[11] K. C.-C. Chang, B. He, C. Li, M. Patel, and Z. Zhang. Structured databases on the web: Observations and implications. *SIGMOD Record*, 33(3):61–70, Sept. 2004.

[12] K. C.-C. Chang, B. He, C. Li, and Z. Zhang. Structured databases on the web: Observations and implications. Technical Report UIUCDCS-R-2003-2321, Dept. of Computer Science, UIUC, Feb. 2003.

[13] V. Crescenzi, G. Mecca, and P. Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *VLDB Conference*, pages 109–118, 2001.

[14] A. Doan, P. Domingos, and A. Y. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *SIGMOD Conference*, 2001.

[15] D. Florescu, A. Y. Levy, and A. O. Mendelzon. Database techniques for the world-wide web: A survey. *SIGMOD Record*, 27(3):59–74, 1998.

[16] GNU. wget. Accessible at `"http://www.gnu.org/-software/wget/wget.html"`.

[17] A. Y. Halevy. Answering queries using views: A survey. *The VLDB Journal*, 10(4):270–294, 2001.

[18] B. He and K. C.-C. Chang. Statistical schema matching across web query interfaces. In *SIGMOD Conference*, 2003.

[19] B. He and K. C.-C. Chang. Making holistic schema matching robust: An ensemble framework with sampling and voting. Technical Report UIUCDCS-R-2004-2451, Dept. of Computer Science, UIUC, July 2004.

[20] B. He, K. C.-C. Chang, and J. Han. Discovering complex matchings across web query interfaces: A correlation mining approach. In *SIGKDD Conference*, 2004.

[21] B. He, T. Tao, and K. C.-C. Chang. Organizing structured web sources by query schemas: A clustering approach. In *CIKM Conference*, 2004.

[22] B. He, Z. Zhang, and K. C.-C. Chang. Knocking the door to the deep web: Integrating web query interfaces. In *SIGMOD Conference, System Demonstration*, 2004.

[23] H. He, W. Meng, C. T. Yu, and Z. Wu. Wise-integrator: An automatic integrator of web search interfaces for e-commerce. In *VLDB Conference*, pages 357–368, 2003.

[24] N. Kushmerick, D. S. Weld, and R. B. Doorenbos. Wrapper induction for information extraction. In *Intl. Joint Conference on Artificial Intelligence (IJCAI)*, pages 729–737, 1997.

[25] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying heterogeneous information sources using source descriptions. In *VLDB Conference*, 1996.

[26] J. Madhavan, P. A. Bernstein, A. Doan, and A. Y. Halevy. Corpus-based schema matching. In *ICDE Conference*, 2005.

[27] J. Madhavan, P. A. Bernstein, P. Domingos, and A. Y. Halevy. Representing and reasoning about mappings between domain models. In *8th national conference on artificial intelligence*, 2002.

[28] J. Madhavan, P. A. Bernstein, and E. Rahm. Generic schema matching with cupid. In *VLDB Conference*, 2001.

[29] K. Marriott. Constraint multiset grammars. In *Proceedings of IEEE Symposium on Visual Languages*, pages 118–125, 1994.

[30] R. J. Miller, M. A. Hernández, L. M. Haas, L. Yan, C. T. Howard Ho, R. Fagin, and L. Popa. The Clio project: managing heterogeneity. *SIGMOD Rec.*, 30(1):78–83, 2001.

[31] Y. Papakonstantinou, H. García-Molina, and J. Ullman. Medmaker: A mediation system based on declarative specifications. In *ICDE Conference*, 1996.

[32] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4):334–350, 2001.

[33] J. D. Ullman. Information integration using logical views. In *ICDT Conference*, Jan. 1997.

[34] W. Wu, C. T. Yu, A. Doan, and W. Meng. An interactive clustering-based approach to integrating source query interfaces on the deep web. In *SIGMOD Conference*, 2004.

[35] L. L. Yan, R. J. Miller, L. M. Haas, and R. Fagin. Data-driven understanding and refinement of schema mappings. In *SIGMOD Conference*, 2001.

[36] 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.

[37] Z. Zhang, B. He, and K. C.-C. Chang. Understanding web query interfaces: Best effort parsing with hidden syntax. In *SIGMOD Conference*, 2004.