BUILDING MASHUPS BY EXAMPLE

by

Rattapoom Tuchinda

A Dissertation Presented to the
FACULTY OF THE GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(COMPUTER SCIENCE)

December 2008

Dedication

To my parents

For their support in this long endeavor.

Acknowledgements

I would like to thank my thesis advisor Craig Knoblock. Craig is a wonderful advisor who always makes an effort to allocate time to all of his student. I have learned from Craig not only how to do research, but also how to be a good presenter and how to manage people and projects. Often times I got stuck and did not make enough progress, but Craig still believed in me and encouraged me to look at different paths. I am grateful for his dedication as an advisor.

I am also grateful for help by Pedro Szekely, my thesis co-chair. Pedro usually suggested insightful solutions whenever I faced difficulties in tackling hard problems. He is almost always available to help when Craig is too busy, and a meeting between Craig, Pedro, and me always means progress on my thesis.

I would also like to thank the rest of the members of my thesis committee: Dr. Sven Koenig and Dr. Daniel O'Leary. I appreciate their comments, suggestions, and support. I also would like to thank Dr. Yolanda Gil. Even though she cannot be on my committe due to a schedule conflict, her insight and suggestions on the user evaluation helped make my experiments go smoothly.

Thanks to my officemates: Snehal Thakkar, Martin Michalowski, and Matt Michelson. They are always there when I call for help. We also have a great time doing no work and

playing FIFA on playstation 2. I am also very much indebted to Phyllis O'Neil for her editing skill and to Alma Nava for her administrative help during my years at ISI.

To my cat, Namkang. Her being there with me since the Master degree helps me get through hard times. Finally, to everyone in my family, epecially Mom. Those long phone calls have kept me going and kept me from giving up.

This research is based upon work supported in part by the National Science Foundation under award number IIS-0324955, in part by the Air Force Office of Scientific Research under grant number FA9550-07-1-0416, and in part by the Defense Advanced Research Projects Agency (DARPA) under Contract No. FA8750-07-D-0185/0004.

Table of Contents

List Of Tables

List Of Figures

Abstract

Accurately integrating the information available on the Internet can provide valuable insights useful in decision making. However, the information one needs is usually scattered among multiple web sites. It can be time-consuming to access, clean, combine, and make sense of that data manually. The latest generation of WWW tools and services enables web users to generate web applications that combine content from multiple sources and provide them as unique services that suit their individual needs. This type of web applications is referred to as a Mashup.

To create Mashups, integration systems must be able to walk users through five separate problems: data extraction, source modeling, data cleaning, data integration, and data display. While there exist attempts to facilitate the process of building information integration applications, none is sufficiently easy to use to enable a web user to build an end-to-end information integration application. As a result, a casual user is put off by the time, effort, and expertise needed to build a Mashup.

In this thesis, I make three contributions that address the problem of building Mashups so that a casual user can easily build one efficiently without having to know any programming language. The first is a consolidated approach that uses a database to link Mashup building problems together. Under this approach, the solution from solving a problem in one area will be leveraged to solve a problem in the next area. The second is a table paradigm for building Mashups, where a user incrementally builds a Mashup by filling table cells with examples instead of specifying programming operations. The third is a query formulation technique that uses constraints to help users build complicated queries by specifying examples.

To validate the approach in this thesis, I have done an extensive evaluation involving more than twenty subjects. The evaluation compared the performace of Karma, the system implementation of my approach, against current state-of-the-art systems: Dapper and Pipes. The result shows that Karma is at least three times faster than Dapper/Pipes on three representative Mashup building tasks. In addition, Karma also allows users to build Mashups that they fail to build using the other systems.

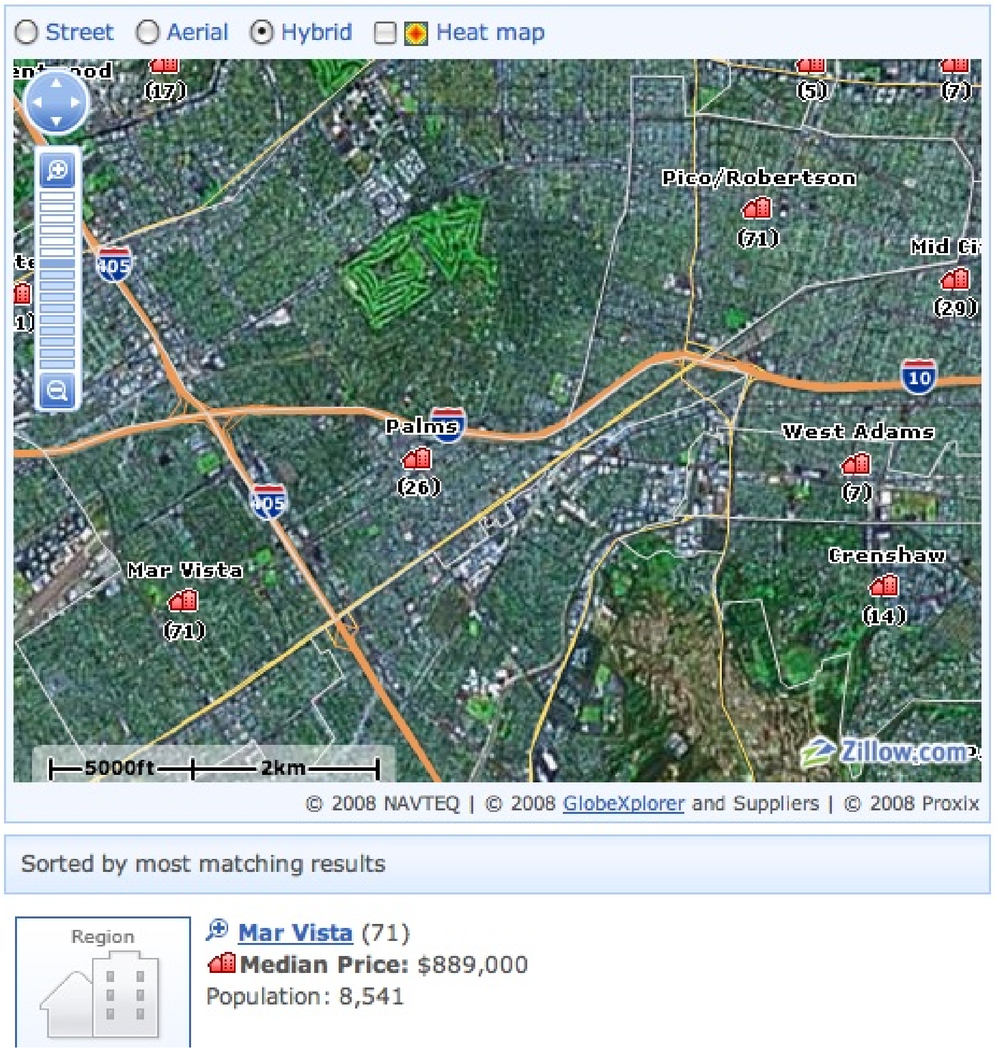
Chapter 1

Introduction

1.1 Motivation and Problem Statement

We need information to make good decisions. In the past, access to necessary information was limited to traditional printed media or a word of mouth. The Internet, however, has changed the information landscape. Nowadays information can be accessed with a click of a mouse. Examples of such information include pricing and reviews of goods and services from multiple vendors, maps, and statistics. Accurately integrating the information available on the Internet can provide valuable insights useful in decision-making. However, the information we need is usually scattered among multiple web sites. It can be time-consuming to access, clean, combine, and try to make sense of that data manually.

For computer-literate users, who can comfortably use computers but are not trained in programming, choices are limited to a) finding the information on their own by browsing multiple web sites, or b) relying on data integration providers to supply web interfaces to access the integrated information. Figure 1.1 shows one such example.



Mar Vista (71)
Median Price: $889,000
Population: 8,541

Figure 1.1: An example Mashup called Zillow. Zillow combines real estate listing web sources with tax property sources. For each house on sale, a Zillow user can see both the sale price and its yearly tax cost.

Zillow (http://www.zillow.com) is one such example that provides real estate listings to help users conduct research on properties. It offers information about properties for sale (e.g., price, taxes, and maps) by integrating information from real estate data sites and property tax sites. However, the inherent problem in any data integration service is approximating its users' needs; what a service offers might not satisfy the needs of each individual user. In addition, there are comparatively few such information integration providers. For example, a family with older kids might want to buy a house in the area with a low crime rating and a good high school ranking. However, it would be impossible for Zillow to cover every aspect of each individual's needs. Do-it-yourself integration systems that can be used by individual users are needed.

A Mashup is a term used to describe a web application that integrates information from multiple data sources. To create Mashups, integration systems must be able to walk users through five separate problems:

- **Data retrieval:** This problem concerns data extraction from a web site. While the concept of Semantic Web has received significant attention recently, most web sites still require a wrapper, an agent that uses information extraction techniques, to convert the data from HTML into structured form.

- **Source modeling:** After extracting data, we need to find out the mapping between new data source and existing data sources. The mapping can be done by assigning a new data column with an existing attribute name when the semantic is the same.

- **Data cleaning:** Data from multiple sources must be normalized to have the same format. For example, the extracted data, *37 Main Street* might need to be transformed to *37 main st.* to match the naming convention of existing data sources.

- **Data integration:** Assuming that the data is normalized, we can treat the problem of combining the data from multiple sources similarly to the way we combine the data from multiple databases. This process is accomplished by formulating queries to access and integrate data across sources.

- **Data display:** Once we combine the data, we need to present it in a way that is easy to understand. Depending on the type of the data, we might opt to use a table or a map, such as a Google Map.

For example, if we want to build a Mashup with similar functionality to Zillow, the workflow for building this Mashup might look like Figure 1.2. First, we need to extract the data from each web source into a form that can be easily manipulated. Extracting data from a web source can be difficult as the data structure might be irregular, the data we need might span multiple pages, and some web sources might require input (i.e., a form) to retrieve the data. Once we manage to extract the data, we need to align the data from different sources. First, we need to align the sources at the semantic level. For example, data columns that contain the address information should be labeled as *address*. Next, we might also need to align the data itself. For example, the realtor site might list a particular street as *37 Washington Boulevard*, while the property tax might use an abbreviation like *37 washington blvd.* After alignment, we need to figure out a way to combine the data from these two sources. A database join on a common attribute *address* can be one of the ways to combine those sources. At the end of the data integration, what we have is a giant table of data points that can be displayed on a map.

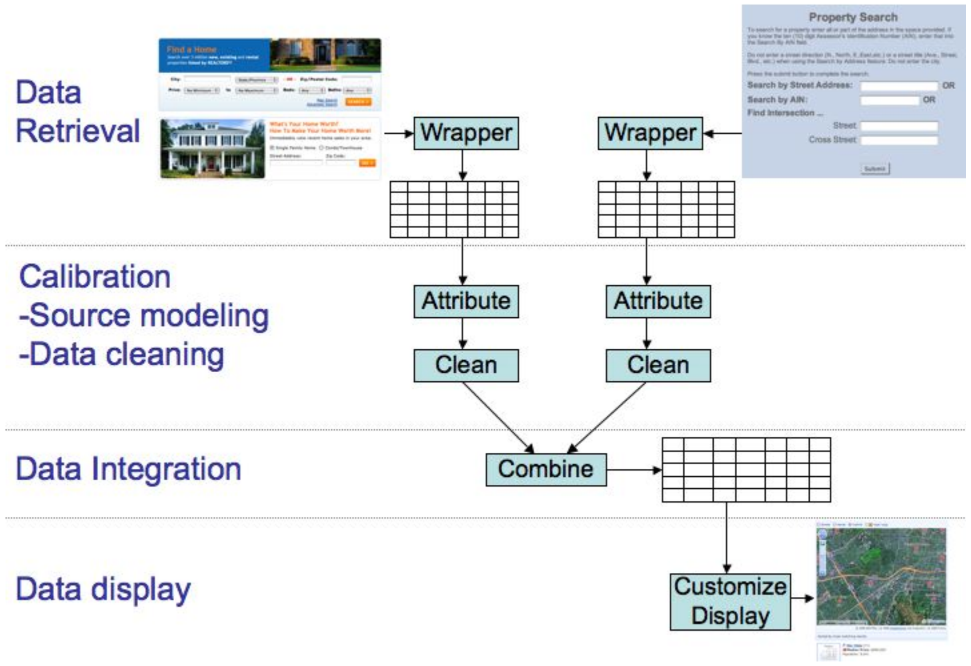
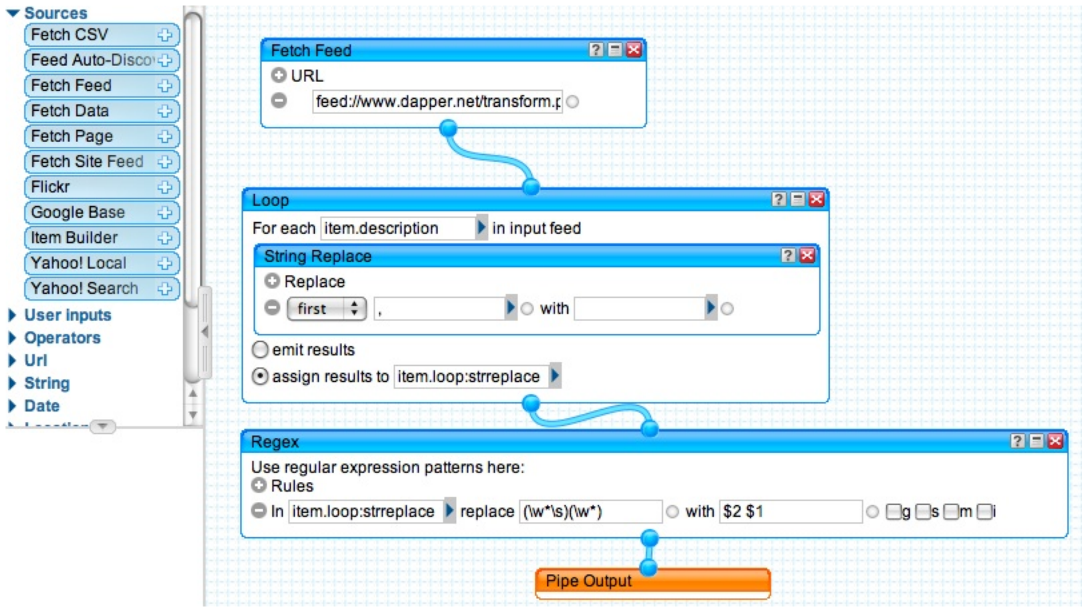


Figure 1.2: A simplified workflow required to build a Mashup similar to Zillow.

While there exist attempts to facilitate the process of building information integration applications, none is sufficiently easy to use to enable a web user to build an end-to-end information integration application. As a result, a casual user is put off by the time, effort, and expertise needed to build a Mashup.

Most existing Mashup tools use what I define as a widget paradigm. Figure 1.3 shows Yahoo's Pipes, a Mashup tool that incorporates the widget paradigm. In the widget paradigm, a user is presented with an array of widgets to choose from, where each one represents a specific programming operation. To build a Mashup, the user needs to drag widgets onto a canvas, customize each widget, and connect them to form a workflow that generates the output data. This output data can then be exported and displayed as an XML, a RSS Feed, or a map.



Fetch Feed
URL
feed://www.dapper.net/transform.p

Loop
For each item.description in input feed
String Replace
Replace
first , with
emit results
[✓ assign results to] item.loop:strreplace

Regex
Use regular expression patterns here:
Rules
In item.loop:strreplace replace (\w*\s)(\w*) with $2 $1 g s m i

Figure 1.3: An example of the widget approach to building Mashups. The user needs to locate, customize, and connect widgets to create a Mashup.

There are three inherent problems with this widget paradigm. First, more programming operations mean more widgets. Locating a specific widget can be time consuming. For example, Yahoo's Pipes has 43 widgets, while Microsoft's Popfly has around 300 widgets. Second, while there is no code to write when using widgets, the fundamental approach behind this paradigm is to abstract a particular programming operation into a widget. As a result, customizing some widgets may require the knowledge of programming. Third, most Mashup tools only address some Mashup building problems while ignoring the others. For example, Dapper mainly addresses data retrieval, while Yahoo's Pipes completely ignores data retrieval, but focuses on data cleaning and data integration; walking their users through all five Mashup building problems seem to be too difficult and time consuming. One of the reasons might be the lack of a paradigm that can encapsulate all these problems into one simple interaction framework. As a result, building Mashups is complicated and the range of Mashups that can be built by naive users is limited.

The research problem that I aim to address in my thesis is to reduce the complexity of building Mashups so that any web user without any programming knowledge can build Mashups.

1.2 Proposed Approach

To support a web user who does not know programming, there are three important features to keep in mind:

- Make it easy: The Mashup building process should not require the user to write code or understand any programming concepts.
- Make it short: The time required to build Mashups should be relatively short.
- Make it intuitive: The interaction platform should be intuitive to the user.

The approach in this thesis is called the Karma framework. This framework aims to realize the above three goals using the following three key ideas:

- Focus on the data, not the operation: For a user with no programming knowledge, interacting with data is more natural. For example, the user might not know which operations/widgets to use to clean *Jones, Norah*, but the user can specify the end result that he/she wants to see in the data form (i.e., *Norah Jones*); the operation is then induced indirectly based on what the user supplies during the interaction by using data and some predefined transformation.

- Leverage existing databases: Instead of trying to solve every problem from scratch, which requires feedback and expertise from the user, we can use existing databases to help simplify the problem. For example, if a data point extracted from a web is misspelled (e.g., wahington instead of washington), it would be time consuming for the user to manually locate the misspelled data point. We can simplify this problem by comparing new data points with existing data points and alert the user when we find any irregularities with a list of suggestions for replacements.

- Consolidate rather than divide-and-conquer: Divide-and-conquer is often regarded as the golden rule to solve many computer science problems. However, many Mashup building issues are interrelated. On the back end, it is possible to exploit the structure such that solving a problem in one area can help simplify the process of solving a problem in another area. On the front end, all the interactions can occur on a spreadsheet platform where the user solves all the problems by manipulating data in a single table instead of trying to create a workflow by connecting widgets.

1.3 Thesis Statement

Web users can build Mashups effectively using an integrated framework that lets them solve the problems of data extraction, source modeling, data cleaning, and data integration by speciflying examples instead of programming operations.

1.4 Contributions of the Research

The main contribution of the thesis is the framework to build Mashups easily. The Karma framework includes:

- A programming-by-demonstration approach that uses a single table for building a Mashup; a user would incrementally builds Mashups by filling table cells with examples, instead of customizing a workflow of connecting widgets.
- A integrated approach that links data extraction, source modeling, data cleaning, and data integration together. Under this approach, the solution from solving a problem in one area will be leveraged to solve a problem in the next area.
- A query formulation technique that allows users to specify examples to build complicated queries.

1.5 Outline of the Thesis

The rest of this thesis is organized as follows. Chapter 2 discusses the three key ideas of the Karma framework and its architecture in detail. Chapter 3 through 6 shows how these key ideas can be applied to each Mashup building problem (i.e., data retrieval, source modeling, data integration, and data display) respectively under a unified table paradigm. Chapter 7 reviews the related work in each problem area. Chapter 8 lays out two types of evaluation plans and discusses their results. Finally, chapter 9 summarizes my contributions and lists possible directions for future work.

Chapter 2

Building Mashups without Programming

In chapter 1, I presented the five problems of building integrated applications, introduced the term Mashup, and proposed a framework to help users with no programming experience build Mashups easily. This chapter provides a) a more in-depth overview of Mashups, b) an intuition of how the Karma framework solves Mashup building problems, and c) a case study example that will be discussed extensively in the remainder of the thesis.

2.1 Overview of Mashups

2.1.1 The Origin of Mashups

A term Mashup has its roots in the music industry. It is defined as a process of combining vocals and rhythm from two separate songs into a new song. This term has propagated through multiple fields including, music, video, and web applications [1]. While there is no standard definition of a web Mashup, most software developers would agree on a loose definition of it as a web application that combines data from multiple sources to create a new application.

2.1.2 Examples of Useful Mashups

A Mashup might allow its consumers to see the same set of data from a different perspective or combine data from multiple sources to help them solve a particular task without having to manually collect data from multiple web sites. I have included a set of Mashups that I found useful or interesting as examples. More Mashup listings can be found at ProgrammableWeb (http://www.programmableweb.com) and google mapsMania (http://google mapssmania.blogspot.com/)

2.1.2.1 Wii Seeker

URL: http://wii.findnearby.net/

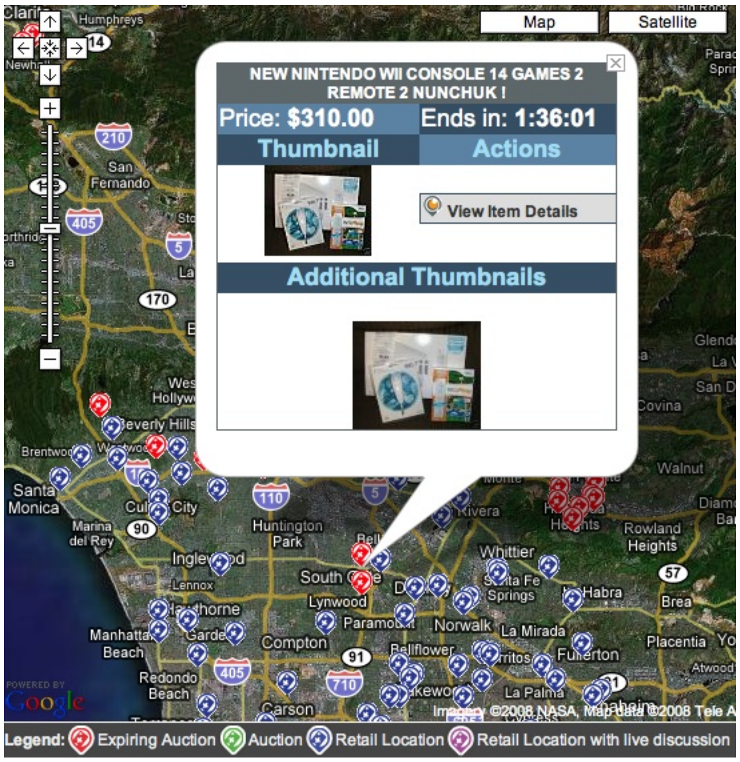
Data sources: eBay, Amazon, Circuit City, Best Buy, and google maps.

Functionality: Wii is a popular gaming console in a very high demand. Shown in Figure 2.1, this Mashup combines a) eBay auction listings b) electronics store listings (i.e., Best Buy, Walmart, and Circuit City) and shows the result on a Google map.

Usefulness: Going to every retailer web site, locating nearby stores, and calling each store to find if it has the console for sale is very time consuming.

2.1.2.2 Ski Bonk

URL: http://www.skibonk.com/ski/index.jsp

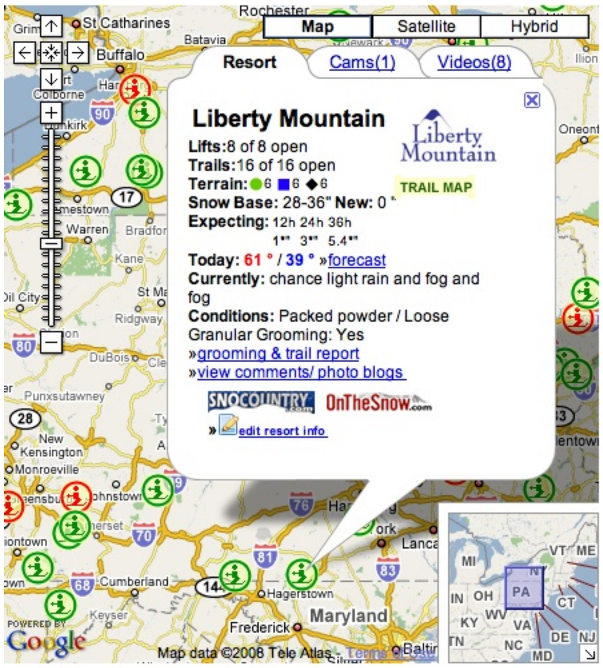

Map
Satellite
Legend: Expiring Auction Auction Retail Location Retail Location with live discussion

NEW NINTENDO WII CONSOLE 14 GAMES 2 REMOTE 2 NUNCHUK !

Price: $310.00	Ends in: 1:36:01
Thumbnail	Actions
	View Item Details

Additional Thumbnails

Figure 2.1: Wii Seeker lists stores and auctions in a nearby area that might have Wii for sale



Liberty Mountain

Lifts: 8 of 8 open
Trails: 16 of 16 open
Terrain: ● 6 ■ 6 ◆ 6
Snow Base: 28-36" **New:** 0 "
Expecting: 12h 24h 36h
1" 3" 5.4"
Today: 61 ° / 39 ° »forecast
Currently: chance light rain and fog and fog
Conditions: Packed powder / Loose Granular Grooming: Yes
»grooming & trail report
»view comments/ photo blogs
» edit resort info

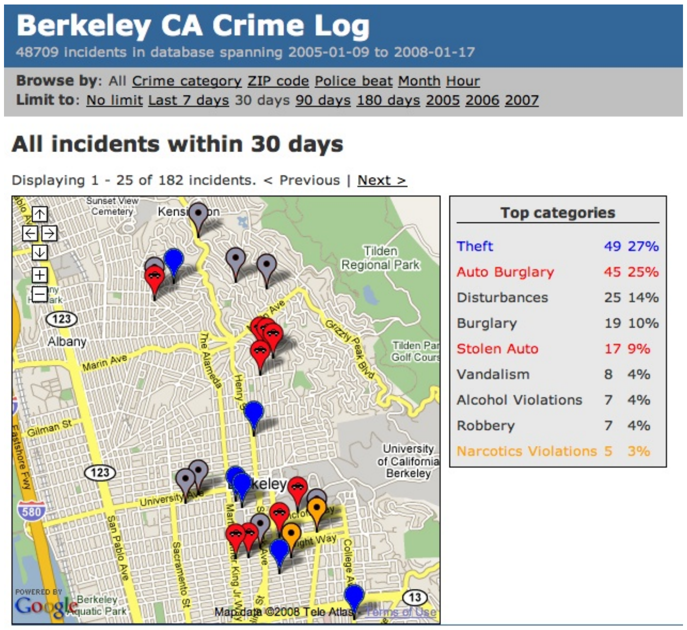
Figure 2.2: SkiBonk combines snow reports with nearby ski resorts.

Data sources: SnoCountry (http://www.snocountry.com/), OnTheSnow (http://www.onthesnow.com/), Weather Underground (http://www.weatherunderground.com/), and the National Weather Service (http://weather.gov/).

Functionality: Shown in Figure 2.2, Ski Bonk combines snow data with resort data and shows the result on a Google Map. A green icon means a particular ski resort has enough snow to be open.

Usefulness: Imagine trying to call every ski resort to find out if it is open, or how many inches of snow it received last night.

2.1.2.3 Berkeley Crimelog



Berkeley CA Crime Log

48709 incidents in database spanning 2005-01-09 to 2008-01-17

Browse by: All Crime category ZIP code Police beat Month Hour
Limit to: No limit Last 7 days 30 days 90 days 180 days 2005 2006 2007

All incidents within 30 days

Displaying 1 - 25 of 182 incidents. < Previous | Next >

Top categories		
Theft	49	27%
Auto Burglary	45	25%
Disturbances	25	14%
Burglary	19	10%
Stolen Auto	17	9%
Vandalism	8	4%
Alcohol Violations	7	4%
Robbery	7	4%
Narcotics Violations	5	3%

Figure 2.3: This Mashup shows the crime information in Berkeley area in the past 30 days.

URL: http://berkeleyca.crimelog.org/all/limit/30days

Data sources: Berkeley police department (http://209.232.44.42/cvc/) and google maps

Functionality: Shown in Figure 2.3, each point on this Mashup represents a crime that happened in the past 30 days. It also provides detailed information about each crime data point (i.e., crime type, date, and time).

Usefulness: Each area is plagued with different types of crimes. A person who lives there or who is planning to live there can better prepare him/herself. For example, if car burglary is prevalent in the area, it might make sense to find an apartment with a garage.

2.1.2.4 Housing Map

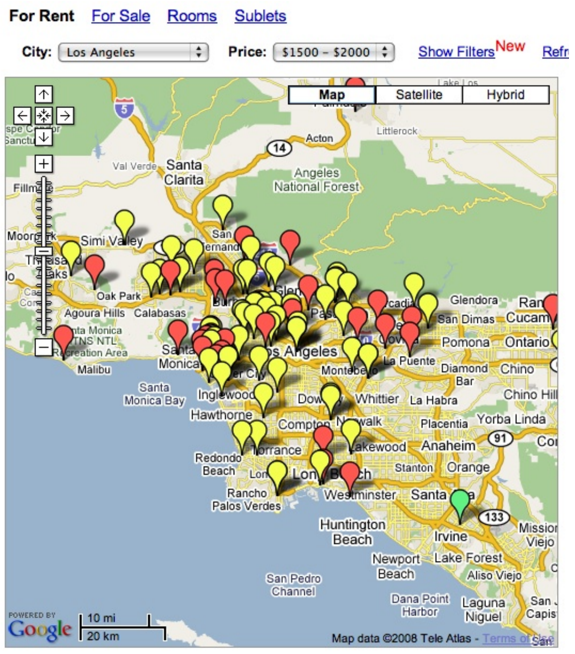


Figure 2.4: This Mashup shows available apartments for rent.

URL: http://www.housingmaps.com/

Data sources: http://www.craigslist.org and google maps

Functionality: Shown in Figure 2.4, this Mashup extracts data from Craiglist apartment posts and puts each rental listing on a Google map.

Usefulness: Craiglist listing does not have a good search infrastructure. As a result, a user looking for an apartment in a particular area would have to look through all the postings. This Mashup allows the user to search for specific listings using criteria such as rental price.

2.1.2.5 Tuneglue

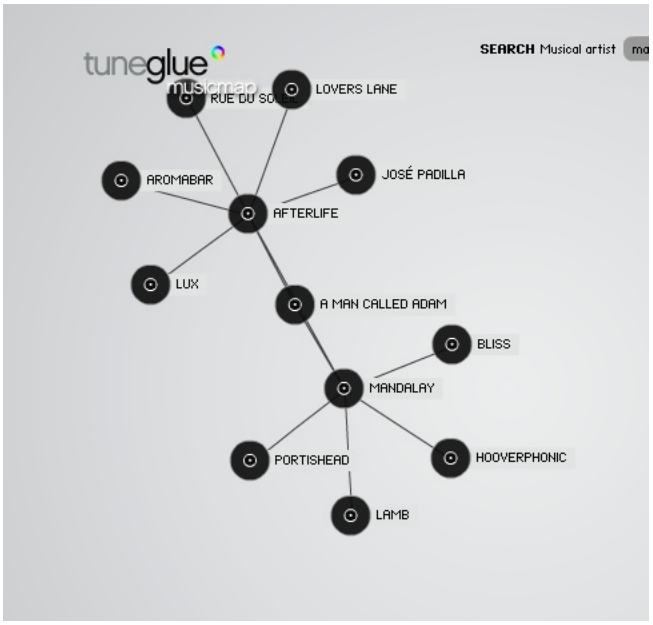


Figure 2.5: This Mashup shows the relationship between artists in terms of their music style.

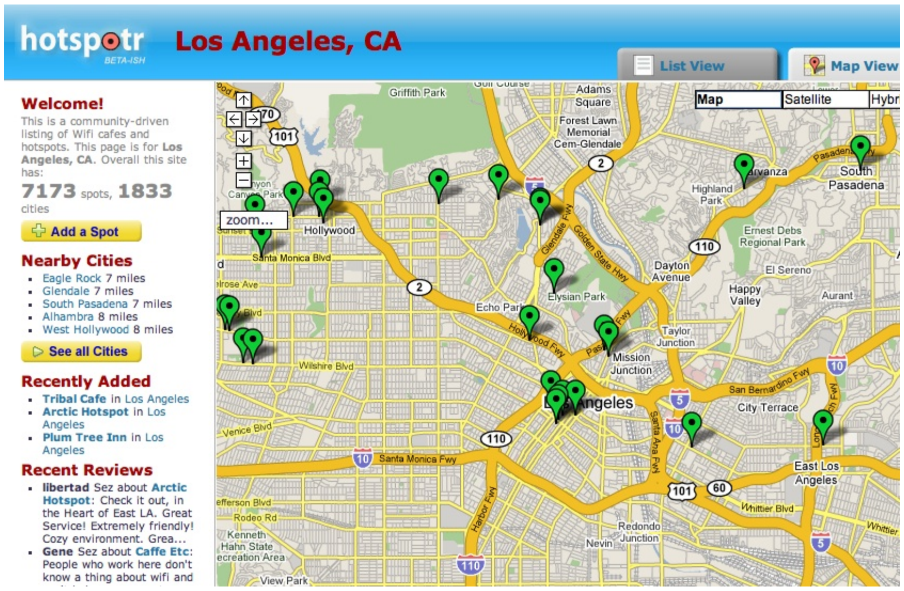
URL: http://audiomap.tuneglue.net/

Data sources: http://www.amazon.com and http://last.fm

Functionality: Shown in Figure 2.5, Tuneglue compiles artist data from last.fm (http://last.fm) and amazon (http://www.amazon.com). By specifying a starting artist, Tuneglue shows related artists based on genre and their musical styles in an explorable graph.

Usefulness: Tuneglue allows its users to expand their musical taste by showing related artists that they might want to check out.

2.1.2.6 Hotspotr



Welcome!
This is a community-driven listing of Wifi cafes and hotspots. This page is for **Los Angeles, CA**. Overall this site has:
7173 spots, **1833** cities

Add a Spot

Nearby Cities
- Eagle Rock 7 miles
- Glendale 7 miles
- South Pasadena 7 miles
- Alhambra 8 miles
- West Hollywood 8 miles

See all Cities

Recently Added
- **Tribal Cafe** in Los Angeles
- **Arctic Hotspot** in Los Angeles
- **Plum Tree Inn** in Los Angeles

Recent Reviews
- **libertad** Sez about **Arctic Hotspot**: Check it out, in the Heart of East LA. Great Service! Extremely friendly! Cozy environment. Grea...
- **Gene** Sez about **Caffe Etc**: People who work here don't know a thing about wifi and

Figure 2.6: This Mashup shows places and areas where wifi is available.

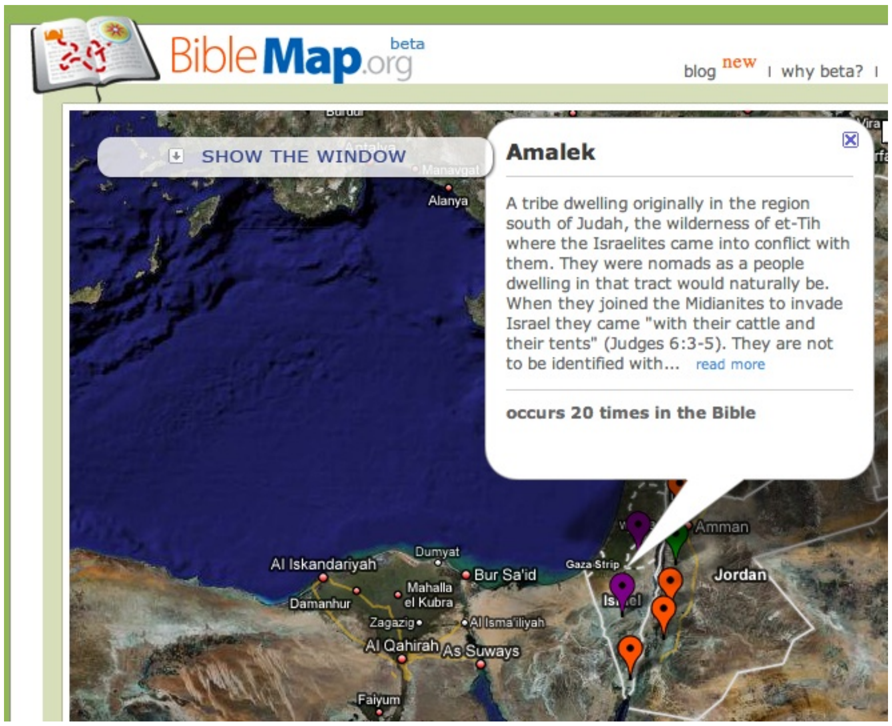
URL: http://hotspotr.com/wifi

Data sources: This Mashup has an interface that allows web surfers to add a new hotspot on the fly into its database. In this case, we treat this database as a single (although dynamic) data source.

Functionality: Shown in Figure 2.6, Hotspotr provides a map of the locations where wifi signal is available.

Usefulness: Wifi hotspots belong to different providers. As a result, browsing for information on a single provider web site will not provide all the wifi spots that are available.

2.1.2.7 BibleMap



Amalek

A tribe dwelling originally in the region south of Judah, the wilderness of et-Tih where the Israelites came into conflict with them. They were nomads as a people dwelling in that tract would naturally be. When they joined the Midianites to invade Israel they came "with their cattle and their tents" (Judges 6:3-5). They are not to be identified with... read more

occurs 20 times in the Bible

Figure 2.7: This Mashup shows places mentioned in the Bible.

URL: http://www.biblemap.org/

Data sources: http://www.biblemapper.com/ and google maps

Functionality: Shown in Figure 2.7, BibleMap shows places mentioned in the Bible on a Google map.

Usefulness: This Mashup could be very useful as a learning tool to help students understand more about the Bible.

2.1.2.8 UNESCO World Heritage Site

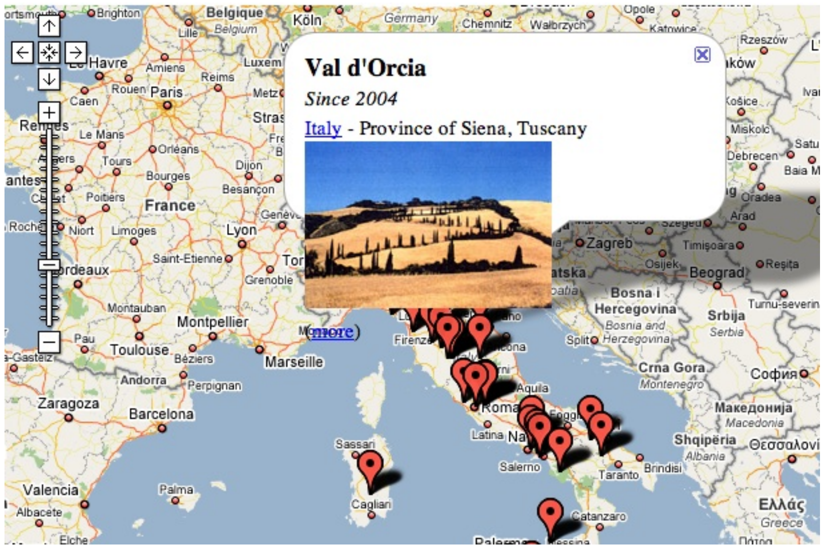


Figure 2.8: This Mashup shows places that have been awarded as UNESCO Heritage sites.

URL: http://labs.beffa.org/worldheritage/

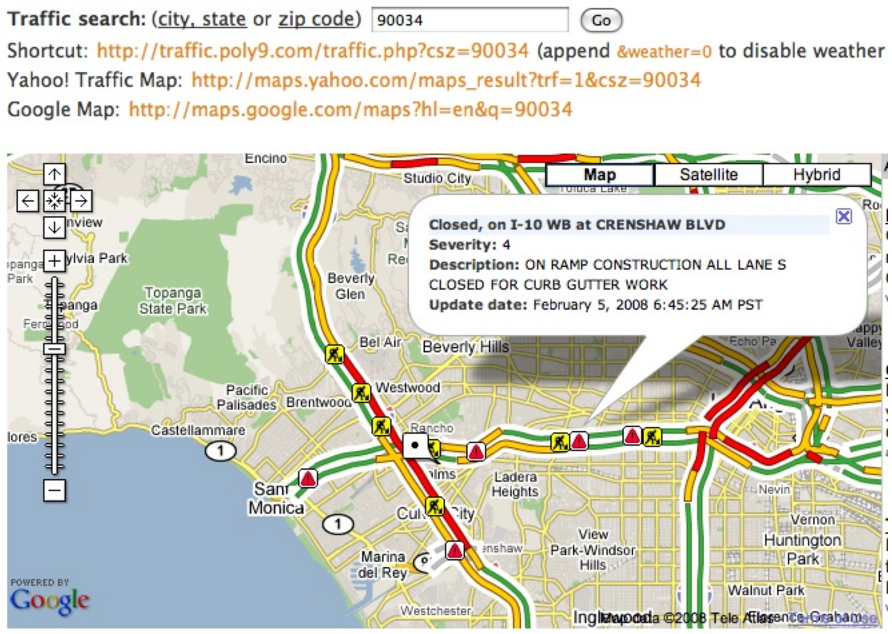
Data sources http://whc.unesco.org/en/list and google maps

Functionality: Shown in Figure 2.8, this Mashup gathers a list of UNESCO Heritage sites and shows them on a map.

Usefulness: This kind of Mashup might be useful when planning a trip to foreign countries.

2.1.2.9 Traffic Map

URL: http://traffic.poly9.com/


Traffic search: (city, state or zip code) 90034 Go
Shortcut: http://traffic.poly9.com/traffic.php?csz=90034 (append &weather=0 to disable weather
Yahoo! Traffic Map: http://maps.yahoo.com/maps_result?trf=1&csz=90034
Google Map: http://maps.google.com/maps?hl=en&q=90034

Closed, on I-10 WB at CRENSHAW BLVD
Severity: 4
Description: ON RAMP CONSTRUCTION ALL LANE S CLOSED FOR CURB GUTTER WORK
Update date: February 5, 2008 6:45:25 AM PST

Figure 2.9: This Mashup shows current accidents and repairs on roads and highways .

Data sources: http://www.yahoo.com and google maps

Functionality: Shown in Figure 2.9, a traffic map Mashup like this is widely available in many US regions. It shows where and when accidents and road repairs occur.

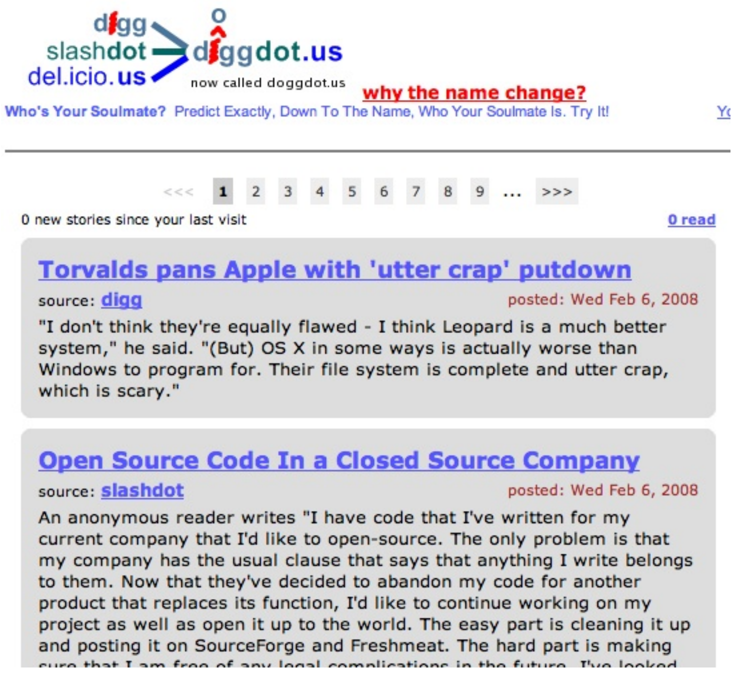
Usefulness: This kind of Mashups is very useful to help avoid traffic congestion when planning a trip.

2.1.2.10 Doggdot

URL: http://doggdot.us/

Data sources: http://www.digg.com, http://www.slashdot.org, and http://del.icio.us

Functionality: Shown in Figure 2.10, Doggdot combines postings from Digg, Slashdot, and Del.icio.us.



0 new stories since your last visit

Torvalds pans Apple with 'utter crap' putdown
source: digg posted: Wed Feb 6, 2008
"I don't think they're equally flawed - I think Leopard is a much better system," he said. "(But) OS X in some ways is actually worse than Windows to program for. Their file system is complete and utter crap, which is scary."

Open Source Code In a Closed Source Company
source: slashdot posted: Wed Feb 6, 2008
An anonymous reader writes "I have code that I've written for my current company that I'd like to open-source. The only problem is that my company has the usual clause that says that anything I write belongs to them. Now that they've decided to abandon my code for another product that replaces its function, I'd like to continue working on my project as well as open it up to the world. The easy part is cleaning it up and posting it on SourceForge and Freshmeat. The hard part is making

Figure 2.10: This Mashup combines postings from famous technology and social networking sites.

Usefulness: Aggregating postings from multiple web sites puts all information in one place, so a user does not have to visit every web site for the information.

2.1.3 Categorizing Mashups

One way to categorize Mashups is by functionality. Figure 2.11 displays Mashup distribution based on functionality.

In this thesis, we categorize Mashups by their workflow structures ranging from the easiest to the hardest to implement. To generate this categorization, we look at the top 50 popular Mashups on the ProgrammableWeb web site and divide them into 5 types of Mashups. While 50 Mashups is a small number, the distribution in terms of functionality is similar to that of Figure 2.11. As a result, we believe this is a good approximation

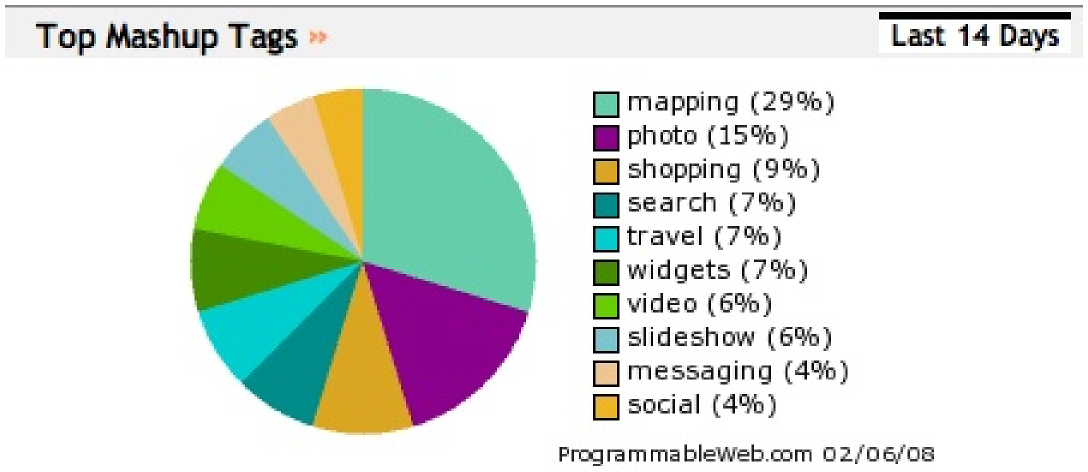


Figure 2.11: The segmentation of Mashups by their functionality from ProgrammableWeb

of the overall Mashup population. The Mashup categorization based on the structure is shown below:

1. **One simple source:** This Mashup type is constructed by extracting data from a single source and placing them on a map. There is minimal data cleaning and no source modeling or data integration. BibleMap, UNESCO World Hertage Site and Berkeley crimelog are examples of this Mashup type.

2. **Combining data points from two or more separate sources:** While this Mashup type incorporates two or more sources, the data does not need to be modeled, cleaned, or integrated. A map is used as a bucket to show the result from each different data source. Housing Map, Wii seeker, Hotspotr, and Doggdot belong to this Mashup type.

3. **One source with form:** To extract the traffic data, the source might require a user to supply the zip code information before the data can be retrieved. As

a result, we need a mechanism to deal with http form and web page navigation. Traffic Map belongs to this Mashup type.

4. **Combining two or more sources using a database join:** The data from one source needs to be joined with data from another source. Skibonk belongs to this Mashup type.

5. **Using specialized operations or displays:** While the underlying data for every Mashup is just a table, this Mashup type uses a specialized display. Tuneglue belongs to this Mashup type.

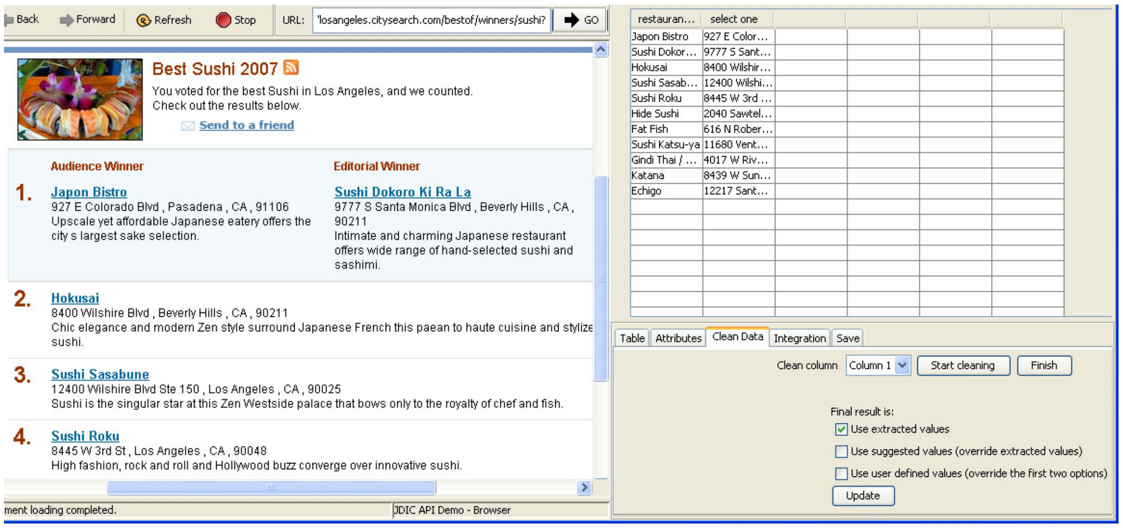
Since the fifth Mashup type requires specialized operations and knowledge, it would not be possible for casual users to implement this Mashup type. As a result, our goal is to be able to easily build the first four types of Mashups, which account for about 47 percent of the most popular Mashups.

2.2 The Programming by Demonstration Framework

Regardless of the categorization method, each Mashup is simply tables of data hiding behind a display. As a result, allowing a user to easily generate these tables of combined data is an important part of building Mashups. This section will give a brief overview of Karma, my software implementation, and elaborate on the three key ideas mentioned earlier in chapter 1.

2.2.1 The Karma System

Figure 2.12 shows Karma, a system that incorporates the programming-by-demonstration framework. The user interface is divided into three areas. First, the left section contains an embedded web browser that allows users to browse the web. Second, the top right section contains the spreadsheet that shows the current data being manipulated. Third, the lower right section contains many modes that users can use to handle each Mashup building problem.



URL: 'losangeles.citysearch.com/bestof/winners/sushi?

Best Sushi 2007

You voted for the best Sushi in Los Angeles, and we counted.
Check out the results below.

Audience Winner

1. Japon Bistro
927 E Colorado Blvd , Pasadena , CA , 91106
Upscale yet affordable Japanese eatery offers the city s largest sake selection.

Editorial Winner

Sushi Dokoro Ki Ra La
9777 S Santa Monica Blvd , Beverly Hills , CA , 90211
Intimate and charming Japanese restaurant offers wide range of hand-selected sushi and sashimi.

2. Hokusai
8400 Wilshire Blvd , Beverly Hills , CA , 90211
Chic elegance and modern Zen style surround Japanese French this paean to haute cuisine and stylize sushi.

3. Sushi Sasabune
12400 Wilshire Blvd Ste 150 , Los Angeles , CA , 90025
Sushi is the singular star at this Zen Westside palace that bows only to the royalty of chef and fish.

4. Sushi Roku
8445 W 3rd St , Los Angeles , CA , 90048
High fashion, rock and roll and Hollywood buzz converge over innovative sushi.

ment loading completed. JDIC API Demo - Browser

restauran...	select one
Japon Bistro	927 E Color...
Sushi Dokor...	9777 S Sant...
Hokusai	8400 Wilshir...
Sushi Sasab...	12400 Wilshi...
Sushi Roku	8445 W 3rd ...
Hide Sushi	2040 Sawtel...
Fat Fish	616 N Rober...
Sushi Katsu-ya	11680 Vent...
Gindi Thai / ...	4017 W Riv...
Katana	8439 W Sun...
Echigo	12217 Sant...

Clean column: Column 1

Final result is:
- [✓] Use extracted values
- [ ] Use suggested values (override extracted values)
- [ ] Use user defined values (override the first two options)

Figure 2.12: The snapshot of Karma.

The spreadsheet section is the center of the interaction. For example, if a user wants to extract the restaurant name from the web page shown in the browser, she would highlight a restaurant name, then drag and drop it onto the spreadsheet. The user would solve each Mashup building problem by manipulating the cells in the table. Finally, when the user is satisfied with the data in the table, she can save the data or choose to display the data on the map, when applicable. From chapter 3 to chapter 6, I will show in detail how each user interaction shapes the data generation on the table.

2.2.2 Programming by Demonstration and Its Focus on Data

One of the key ideas from chapter 1 is the focus on data, not operations. The root of this approach is from the concept called programming-by-demonstration (PBD) [17, 35]. PBD automatically deduces operations to perform repetitive tasks by monitoring how a user interacts with a system. Some examples of a PBD systems include programming by shell script [34], learning procedures for technical support [36], and generating web pages for a mobile platform by browsing [47].

How operations are learned in PBD largely depends on techniques and problem domains. In my thesis, I analyze the structure of the Mashup building problem and generate specific heuristics for each problem area. Figure 2.13 shows an example. To extract a list of data from a web page, a user demonstrates to Karma by dragging a data element from that list and dropping it onto a table.

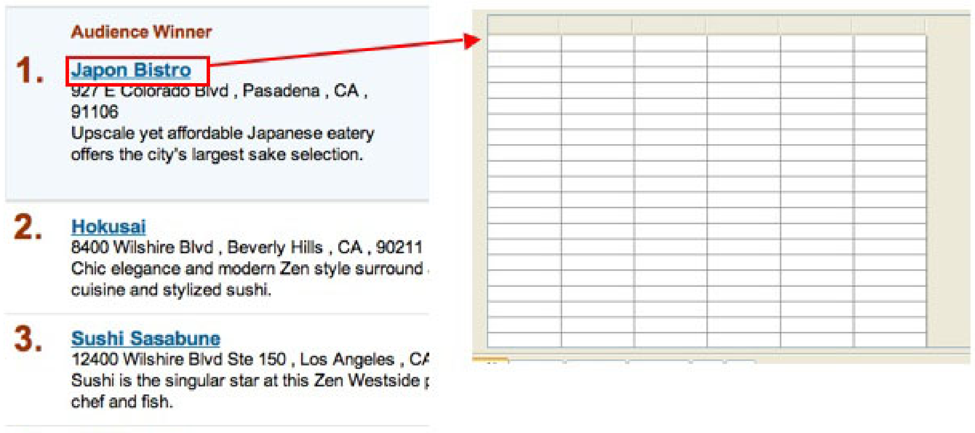

Audience Winner
1.
Japon Bistro
927 E Colorado Blvd , Pasadena , CA , 91106
Upscale yet affordable Japanese eatery offers the city's largest sake selection.
2.
Hokusai
8400 Wilshire Blvd , Beverly Hills , CA , 90211
Chic elegance and modern Zen style surround
cuisine and stylized sushi.
3.
Sushi Sasabune
12400 Wilshire Blvd Ste 150 , Los Angeles , CA
Sushi is the singular star at this Zen Westside
chef and fish.

Figure 2.13: Instead of writing rules to extract the data from a web page, PBD allows users to do it by demonstration through interactions, such as dragging and dropping.

Since that data element belongs to a list based on the page structure, PBD deduces that the user wants to extract the whole list and proceeds to extract the rest of the list.

While this approach has a disadvantage that the defined heuristics might be wrong, a) it is very simple for a user who does not want to spend a lot of times writing operators or training a system, and b) it works well on some domains where the structure of the problem is quite limited. In the Mashup building domain, there are enough structures that can be exploited for this approach to work well. For example, this approach has been widely used in various Mashup building tools (i.e., Dapper and Simile[30]). For those interested in finding out more about how to cope with the problem of over-generalization, under-generalization, or how to recover from errors deduction in PBD, [13] provides a theoretical framework to address those problems.

2.2.3 Leveraging Databases

The idea of leveraging databases to help solve a problem is not unique as it has been used extensively in the data extraction[43], schema matching[18], record linkage[42], data cleaning[43], and data integration[5]. In my thesis, these databases are built with initial set of tables to cover some domains, such as restaurants and cars. The size of these databases grow larger to cover more data and domains as users build more Mashups and save them into the databases. Note that the users do not need to customize the databases such as creating foreign keys because this information is deduced automatically when new tables are added to the databases.

Figure 2.14 shows an example of how to use a database to help solve the source modeling problem. Assuming that there exist the three tables in a database: LA Health Rating, Artist Info, and Zagat. After the user extracts the restaurant data from the previous section, we want to help her assign the appropriate attributes for this new data.

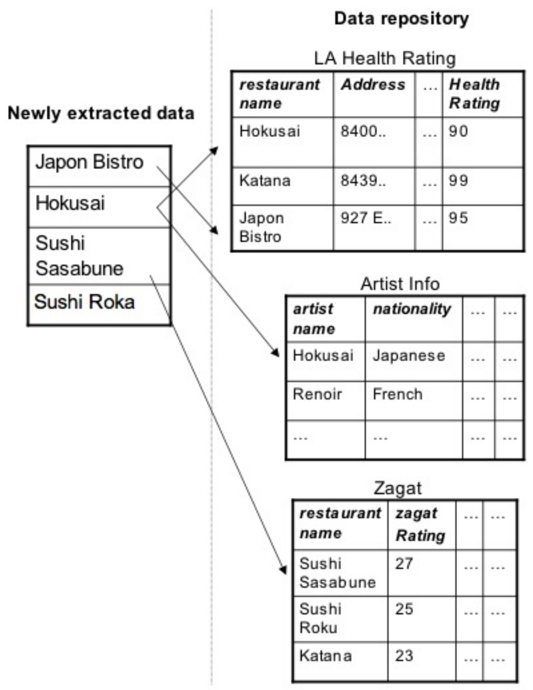


Figure 2.14: By leveraging existing databases, misspelling can be detected automatically.

By looking at how the new data elements overlap with existing data elements, a list of possible candidates is created so the user can choose from these choices. In this example, the new data elements overlap with existing data columns associated with *restaurant name* and *artist name* and the user is presented with those two choices to pick from.

2.2.4 Problem Consolidation

There are separate research areas focusing on data extraction, source modeling, data cleaning, and data integration. While each problem is complex on its own, solving a problem in one area can generate information that can be used to help solving a problem in another area. If the effort is focused to solve one problem at a time, that useful information might be discarded, requiring starting from scratch again when solving another problem.

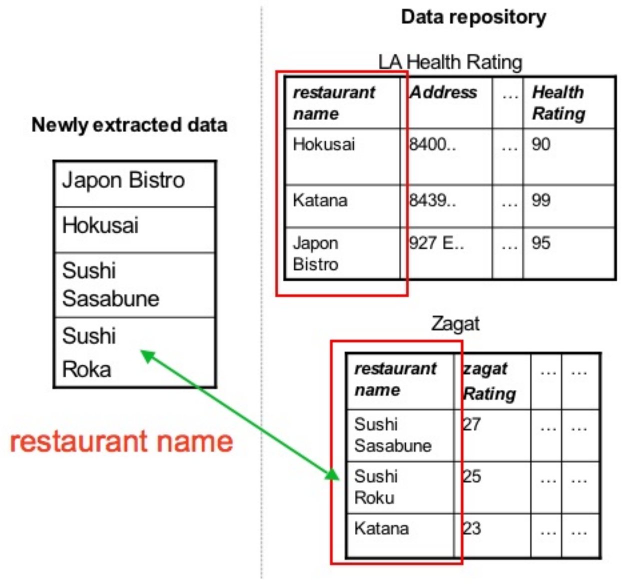


Figure 2.15: As soon as the column name of the newly extracted data is defined during the source modeling mode, we can use this information to help start the cleaning data mode. In this example, a misspell is detected for the entry *Sushi Roka.*

For example, earlier we used the database to help guide the user to select the attribute for the new data column. Once the attribute information for the new data column is known, the data cleaning process can start right away. Since we know that the new column is of type *restaurant name*, we can isolate the data of type *restaurant name* from many tables in the database and apply a string comparison algorithm to detect possible misspelling in the newly acquired data.

2.2.5 The Karma Architecture

The architecture of Karma is shown in Figure 2.16. Its properties are listed below:

- There are two points of entry. A user can start building a Mashup by extracting data from a web page or composing a table from scratch by integrating only data existing in databases.

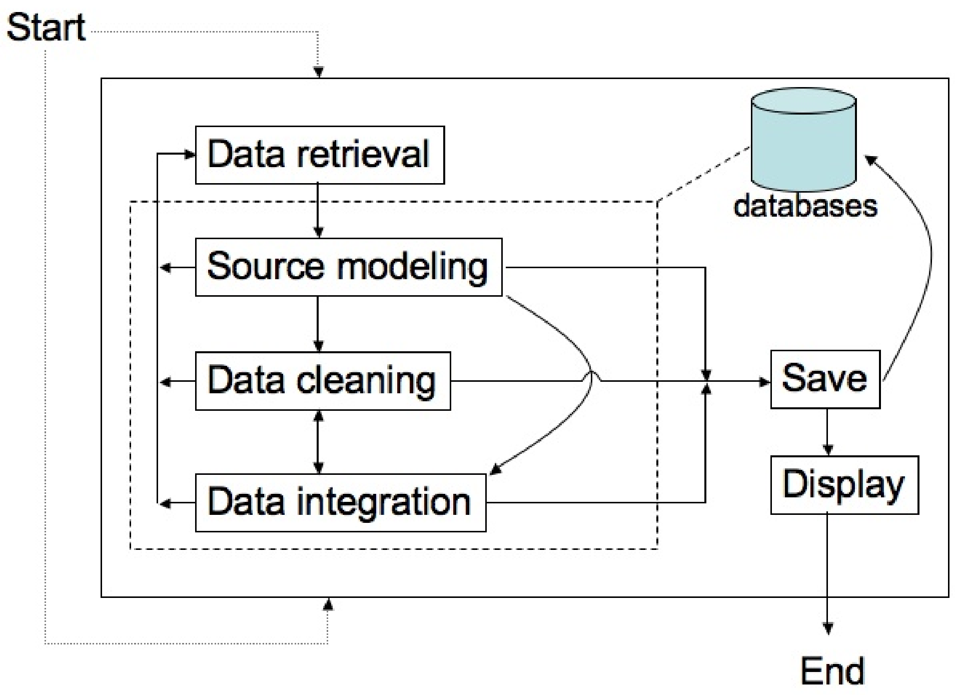


Figure 2.16: The architecture of Karma.

- Source modeling, data cleaning, and data integration make uses of databases to help generate suggestions to a user.
- The user can skip some steps depending on her needs. For example, building a Mashup from one data source might not require data cleaning and data integration.
- The user does not have to finish extracting all the data from the data retrieval right away. For example, three columns of data might be extracted and processed in other problem areas. Then, the user can go back to the same web page and extract more data.
- When the user saves the Mashup, the data is integrated into the databases so it can be a) recalled later, or b) used to help future Mashup building processes.

2.3 Case Study

While it would be nice to have one coherent case study with one example, it would not be possible to showcase the different features and problems that Karma can solve. As a result, I will use multiple examples from chapter 3 to chapter 6. However, at the end of chapter 6, we will have enough examples to construct a Mashup with the structure similar to that in Figure 2.17

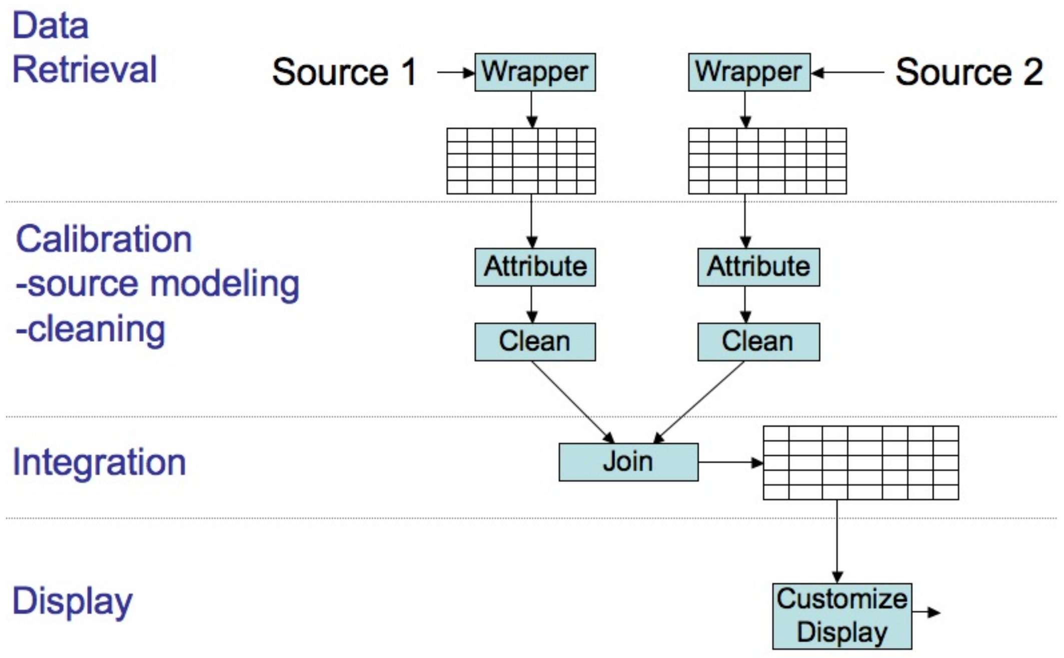


Figure 2.17: The Mashup structure is a superset of all four Mashup types.

Since our aim is to address the four Mashup types described earlier, this Mashup structure can be modified to fit any of those cases; each of the four Mashup types is a subset of this example:

- To build a Mashup that uses only one simple data source, we can focus on the left branch without doing data cleaning or data integration.

- To build a Mashup that combines data points from two or more separate sources without joining, we can ignore the data integration part and proceed to put all the points on the map.
- To build a Mashup from one source with a form, we can focus on one branch and use a source with a form input without doing any data cleaning or data integration.
- To build a Mashup that combines two or more sources using a database join, we need to do all the steps outlined in Figure 2.17.

Starting in the next chapter, we will show how Karma solves each Mashup building problem using different case study examples.

Chapter 3

Data Retrieval

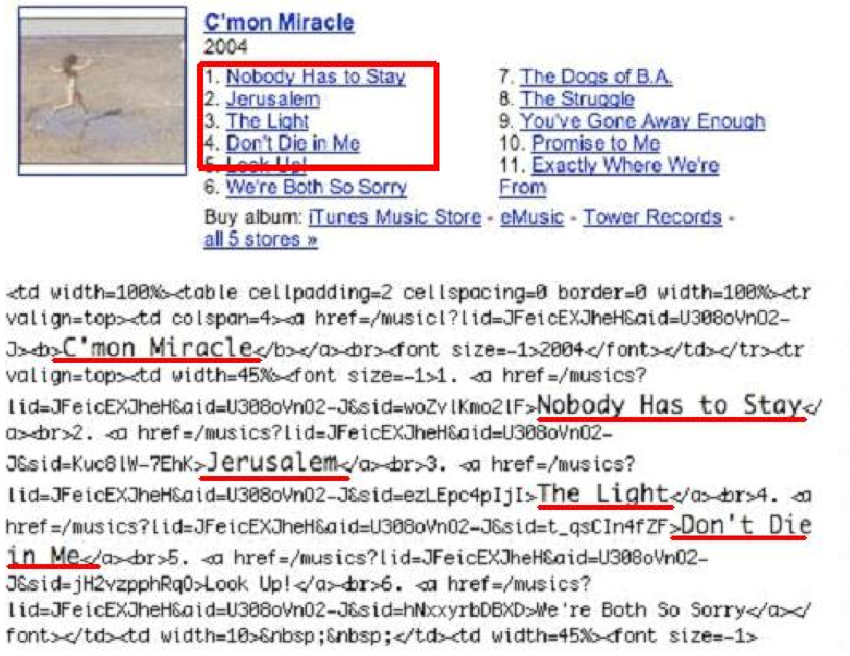
This chapter is divided into three sections. The first section breaks the data retrieval problem down into two subproblems: data extraction and page navigation. The second section first shows how a user would interact with Karma, and describes the technical details behind those interactions. The third section elaborates on how my approach to the data retrieval problem fits the overall Karma framework.

3.1 Problem Definition

A large body of research work usually considers data retrieval as a problem of extracting data from a web page. However, in the real world, data retrieval is composed of two subproblems: data extraction and page navigation.

The problem of data extraction focuses on figuring out how to extract data out of HTML from a specific page or set of similar pages. For example, Figure 3.1 shows a snippet of one of the sources from the case study and its corresponding HTML code.

Page navigation deals with cases where the data we want to extract requires us to go through multiple pages to get to it. For example, each song name in Figure 3.1 has a link to its detail page. One of the detail pages is shown in Figure 3.2. The detail page contains the duration of a specific song that the user might want to extract.


C'mon Miracle
2004

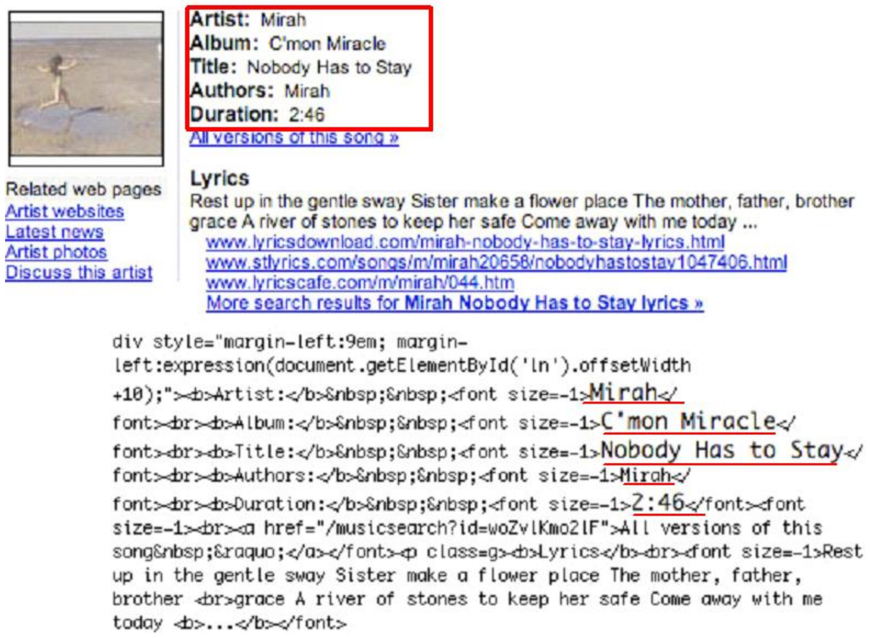
1. Nobody Has to Stay
2. Jerusalem
3. The Light
4. Don't Die in Me
5. Look Up!
6. We're Both So Sorry
7. The Dogs of B.A.
8. The Struggle
9. You've Gone Away Enough
10. Promise to Me
11. Exactly Where We're From

Buy album: iTunes Music Store - eMusic - Tower Records - all 5 stores »

```
<td width=100%><table cellpadding=2 cellspacing=0 border=0 width=100%><tr
valign=top><td colspan=4><a href=/musicl?lid=JFeicEXJheH&aid=U308oVn02-
J><b>C'mon Miracle</b></a><br><font size=-1>2004</font></td></tr><tr
valign=top><td width=45%><font size=-1>1. <a href=/musics?
lid=JFeicEXJheH&aid=U308oVn02-J&sid=woZvlKmo2lF>Nobody Has to Stay</
a><br>2. <a href=/musics?lid=JFeicEXJheH&aid=U308oVn02-
J&sid=Kuc8lW-7EhK>Jerusalem</a><br>3. <a href=/musics?
lid=JFeicEXJheH&aid=U308oVn02-J&sid=ezLEpc4pIjI>The Light</a><br>4. <a
href=/musics?lid=JFeicEXJheH&aid=U308oVn02-J&sid=t_qsCIn4fZF>Don't Die
in Me</a><br>5. <a href=/musics?lid=JFeicEXJheH&aid=U308oVn02-
J&sid=jH2vzpphRq0>Look Up!</a><br>6. <a href=/musics?
lid=JFeicEXJheH&aid=U308oVn02-J&sid=hNxxyrbDBXD>We're Both So Sorry</a></
font></td><td width=10>&nbsp;&nbsp;</td><td width=45%><font size=-1>
```

Figure 3.1: A snippet of the case study source and its corresponding HTML tag. The information that a user wants to extract might be the list of albums and their corresponding song names.

Page navigation is rarely addressed in research because it is considered more or less an engineering issue from the information retrieval point of view. However, from the Mashup building perspective, if we want to make building Mashups easy, we need an approach to do page navigation easily and effectively. To solve the issue of page navigation, we need to figure out how to extract, associate, and combine the data from the first page with its detail pages.


Artist: Mirah
Album: C'mon Miracle
Title: Nobody Has to Stay
Authors: Mirah
Duration: 2:46
All versions of this song »

Related web pages
Artist websites
Latest news
Artist photos
Discuss this artist

Lyrics
Rest up in the gentle sway Sister make a flower place The mother, father, brother grace A river of stones to keep her safe Come away with me today ...
www.lyricsdownload.com/mirah-nobody-has-to-stay-lyrics.html
www.stlyrics.com/songs/m/mirah20658/nobodyhastostay1047406.html
www.lyricscafe.com/m/mirah/044.htm
More search results for Mirah Nobody Has to Stay lyrics »

```
div style="margin-left:9em; margin-
left:expression(document.getElementById('ln').offsetWidth
+10);"><b>Artist:</b>&nbsp;&nbsp;<font size=-1>Mirah</
font><br><b>Album:</b>&nbsp;&nbsp;<font size=-1>C'mon Miracle</
font><br><b>Title:</b>&nbsp;&nbsp;<font size=-1>Nobody Has to Stay</
font><br><b>Authors:</b>&nbsp;&nbsp;<font size=-1>Mirah</
font><br><b>Duration:</b>&nbsp;&nbsp;<font size=-1>2:46</font><font
size=-1><br><a href="/musicsearch?id=woZvlKmo2lF">All versions of this
song&nbsp;&raquo;</a></font><p class=g><b>Lyrics</b><br><font size=-1>Rest
up in the gentle sway Sister make a flower place The mother, father,
brother <br>grace A river of stones to keep her safe Come away with me
today <b>...</b></font>
```

Figure 3.2: A detail page and its corresponding HTML source.

3.2 Approach

In this section, we first define a data retrieval task. Then a snapshot of interactions between a user and Karma is shown. Finally, we highlight technical details that enable those interactions.

3.2.1 Case Study Walkthrough

Let us assume that a user wants to retrieve and combine music data from two particular artists. The structure of web pages is shown in Figure 3.3. For each artist, the album and song listing are on the first page and the information about each specific song (i.e., duration) is in a different detail page. Figure 3.1 shows an example front page for the artist *Mirah*, while Figure 3.2 shows an example detail page for the song *Nobody Has to Stay*.

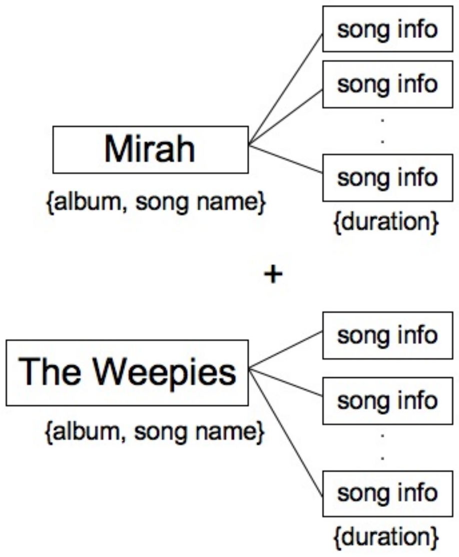
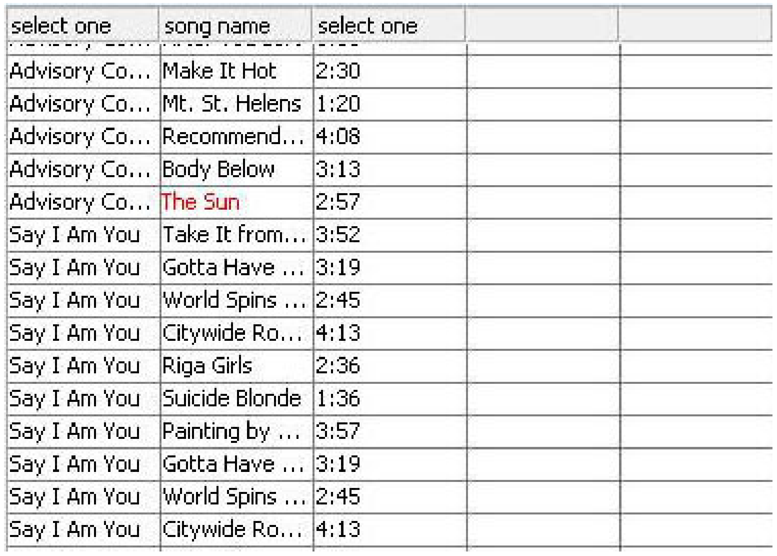


Figure 3.3: Overview of the structure of web pages in a case study task.

The goal is to retrieve the data about *album, song name,* and *duration* into the table shown in Figure 3.4. The step-by-step interaction between the user and Karma is shown from Figure 3.5 to Figure 3.17.

3.2.2 Document Object Model (DOM)

Data retrieval techniques in this chapter mostly rely on a data structure called the Document Object Model (DOM). W3C defines the DOM as "a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents. The document can be further processed and the results of that processing can be incorporated back into the presented page." Currently, existing web browsers convert each HTML page into a DOM before rendering it as a web page. As a result, DOM is easily accessible through existing programming libraries.



select one	song name	select one		
Advisory Co...	Make It Hot	2:30		
Advisory Co...	Mt. St. Helens	1:20		
Advisory Co...	Recommend...	4:08		
Advisory Co...	Body Below	3:13		
Advisory Co...	The Sun	2:57		
Say I Am You	Take It from...	3:52		
Say I Am You	Gotta Have ...	3:19		
Say I Am You	World Spins ...	2:45		
Say I Am You	Citywide Ro...	4:13		
Say I Am You	Riga Girls	2:36		
Say I Am You	Suicide Blonde	1:36		
Say I Am You	Painting by ...	3:57		
Say I Am You	Gotta Have ...	3:19		
Say I Am You	World Spins ...	2:45		
Say I Am You	Citywide Ro...	4:13		

Figure 3.4: The goal for the data retrieval task is to have a table that contains album name, song name, and duration for each song from both artists.

The DOM is also used in data retrieval tools such as simile[30] and dapper. In this subsection, I will cover the basic DOM approach to extracting list data from a web page. Figure 3.18 shows a simplified DOM representation of the HTML snippet shown in Figure 3.1. Every HTML page can be represented using a DOM tree structure.

When a user highlight-drag-drops particular text from a web page to a table, we can identify the location of a particular node in that web page's DOM tree. In our example, when the user extracts *C'mon Miracle*, the location of the data is: *\html\table[1]\tr[1]\td\-a\b*

This location is represented using an XPath expression. XPath is a language used to process tree-like XML documents. In our example, the XPath above means that to get to the data: first take the *html* tag, pick the first *table* tag, pick the first *tr* tag, and continue down *td*, *a*, and *b* tag to find the data.

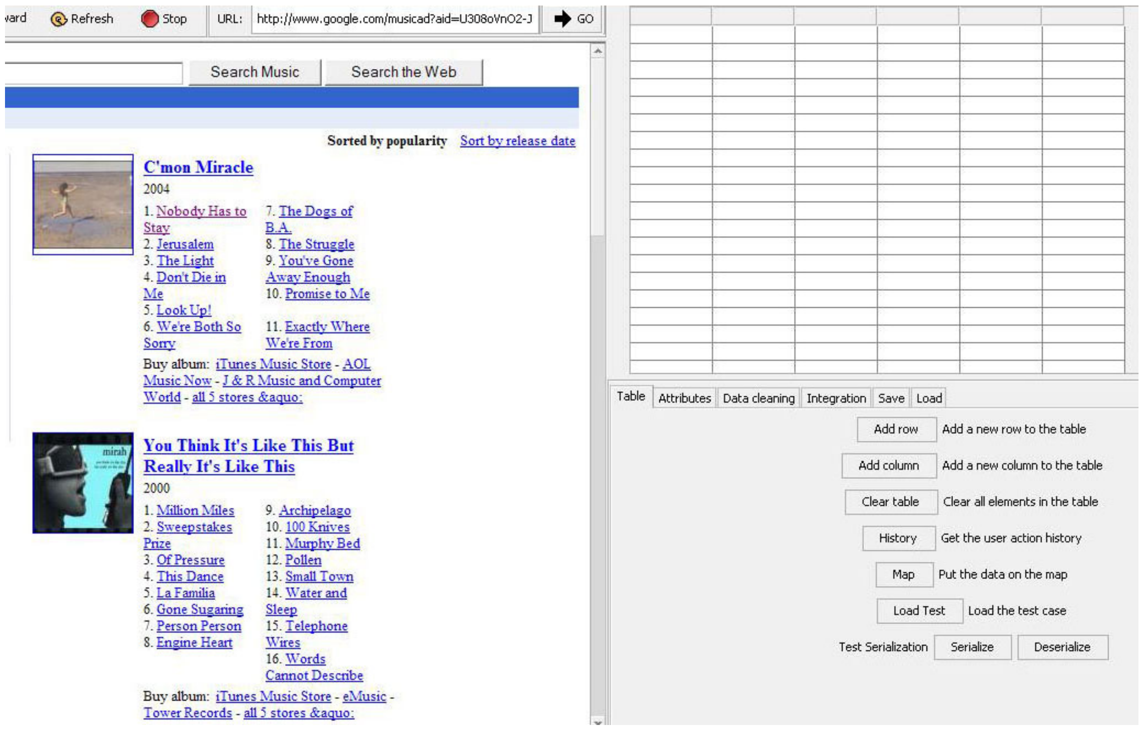


Figure 3.5: Step 1: The user types the URL in the embedded browser to go to a particular page.

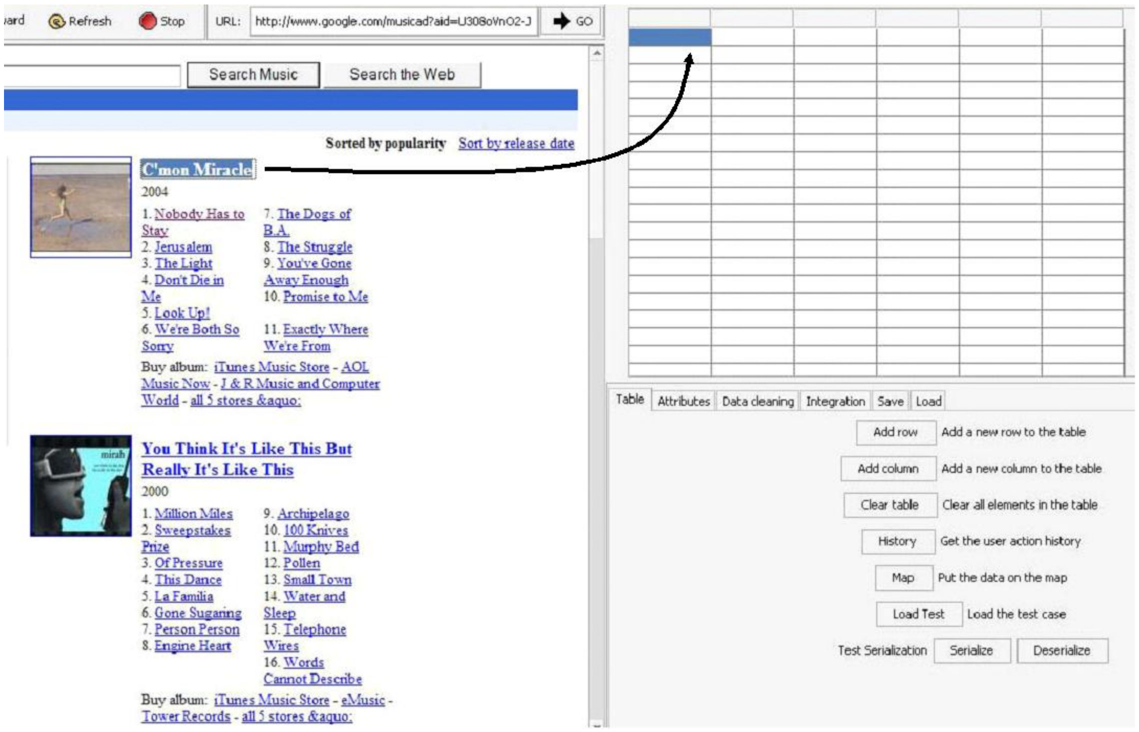
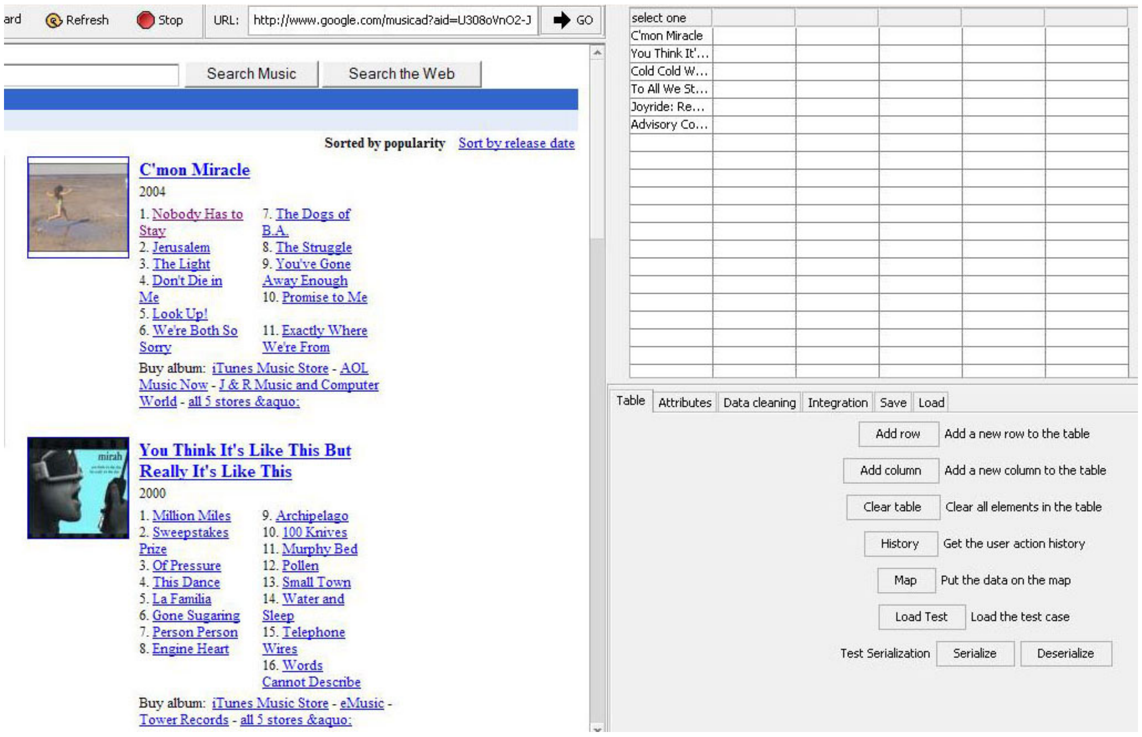

URL: http://www.google.com/musicad?aid=U3O8oVnO2-J

Figure 3.6: Step 2: To extract an album name, the user highlights the data on the page (*C'mon Miracle*), drags the data, then drops it into the cell (0,0) of the table.



http://www.google.com/musicad?aid=U308oVnO2-J

Sorted by popularity Sort by release date

C'mon Miracle
2004
1. Nobody Has to Stay
2. Jerusalem
3. The Light
4. Don't Die in Me
5. Look Up!
6. We're Both So Sorry
7. The Dogs of B.A.
8. The Struggle
9. You've Gone Away Enough
10. Promise to Me
11. Exactly Where We're From
Buy album: iTunes Music Store - AOL Music Now - J & R Music and Computer World - all 5 stores &aquo;

You Think It's Like This But Really It's Like This
2000
1. Million Miles
2. Sweepstakes Prize
3. Of Pressure
4. This Dance
5. La Familia
6. Gone Sugaring
7. Person Person
8. Engine Heart
9. Archipelago
10. 100 Knives
11. Murphy Bed
12. Pollen
13. Small Town
14. Water and Sleep
15. Telephone Wires
16. Words Cannot Describe
Buy album: iTunes Music Store - eMusic - Tower Records - all 5 stores &aquo;

select one					
C'mon Miracle					
You Think It'...					
Cold Cold W...					
To All We St...					
Joyride: Re...					
Advisory Co...					

Add row — Add a new row to the table
Add column — Add a new column to the table
Clear table — Clear all elements in the table
History — Get the user action history
Map — Put the data on the map
Load Test — Load the test case

Figure 3.7: Step 3: Karma recognizes that *C'mon Miracle* is a member of a list and proceeds to extract the rest of the list and fill in the table under the same column.

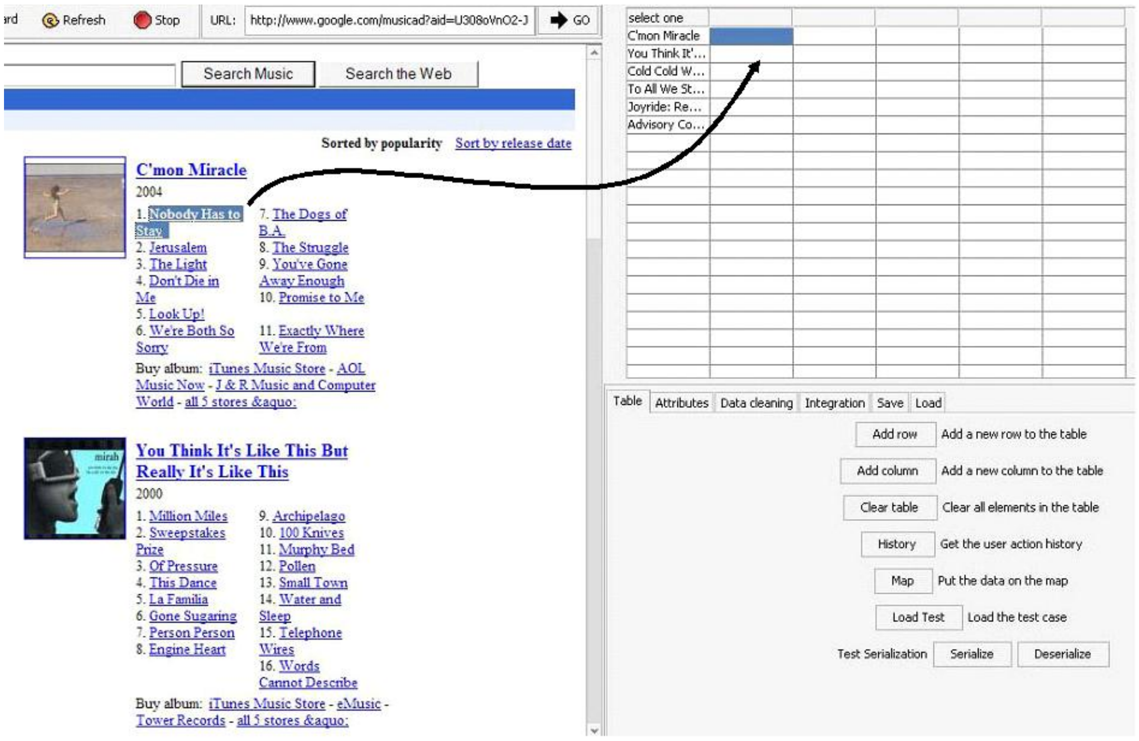
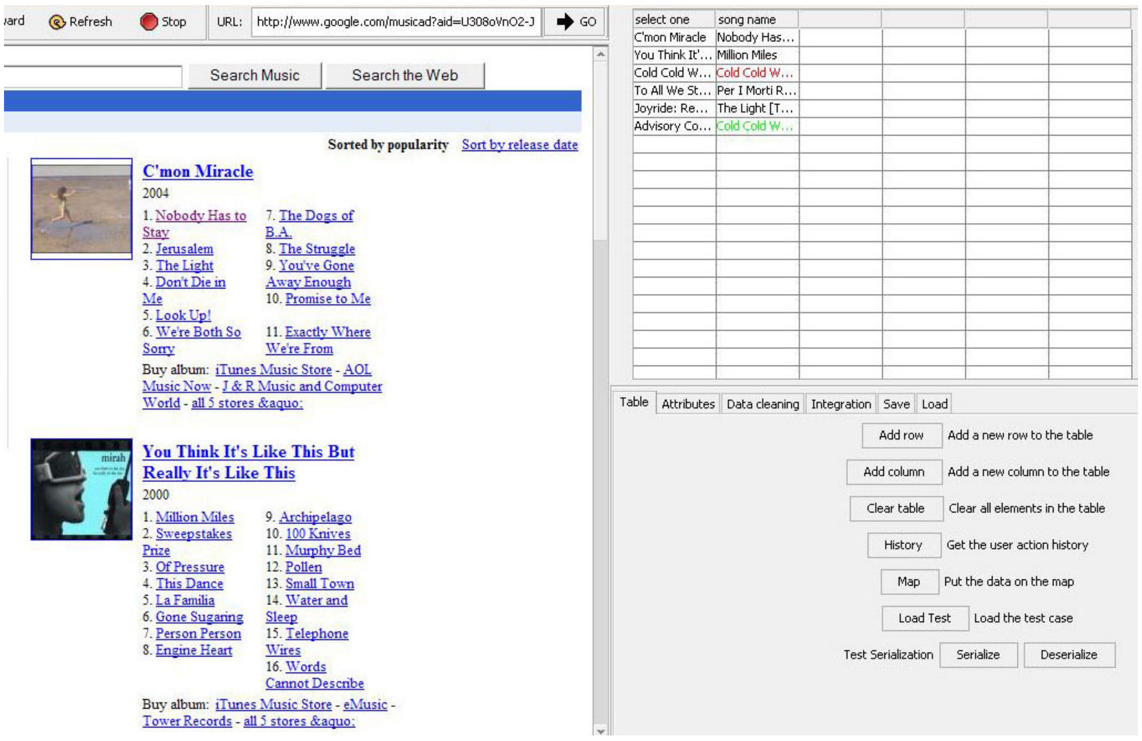


Figure 3.8: Step 4: The user then proceeds to extract the first song from the first album and drop it into the cell (0,1) of the table next to the first album name.



Sorted by popularity Sort by release date

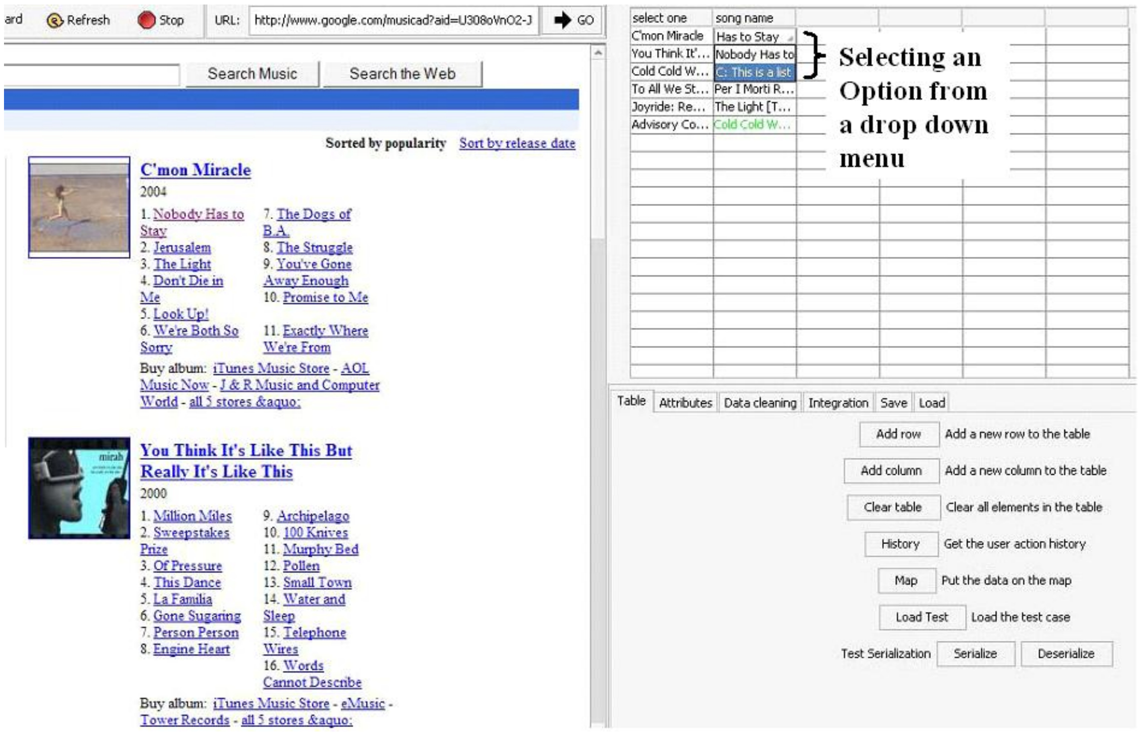
C'mon Miracle
2004
1. Nobody Has to Stay
2. Jerusalem
3. The Light
4. Don't Die in Me
5. Look Up!
6. We're Both So Sorry
7. The Dogs of B.A.
8. The Struggle
9. You've Gone Away Enough
10. Promise to Me
11. Exactly Where We're From
Buy album: iTunes Music Store - AOL Music Now - J & R Music and Computer World - all 5 stores &aquo;

You Think It's Like This But Really It's Like This
2000
1. Million Miles
2. Sweepstakes Prize
3. Of Pressure
4. This Dance
5. La Familia
6. Gone Sugaring
7. Person Person
8. Engine Heart
9. Archipelago
10. 100 Knives
11. Murphy Bed
12. Pollen
13. Small Town
14. Water and Sleep
15. Telephone Wires
16. Words Cannot Describe
Buy album: iTunes Music Store - eMusic - Tower Records - all 5 stores &aquo;

select one	song name				
C'mon Miracle	Nobody Has…				
You Think It'…	Million Miles				
Cold Cold W…	Cold Cold W…				
To All We St…	Per I Morti R…				
Joyride: Re…	The Light [T…				
Advisory Co…	Cold Cold W…				

Add a new row to the table
Add a new column to the table
Clear all elements in the table
Get the user action history
Put the data on the map
Load the test case

Figure 3.9: Step 5: Karma proceeds to extract and fill the first song from every album. Note that the attribute name in the second column is set to *song name*, while the one in the first column is set to *select one*. The explanation of these behaviors, including light color fonts for some data elements, will be explained in later chapters.

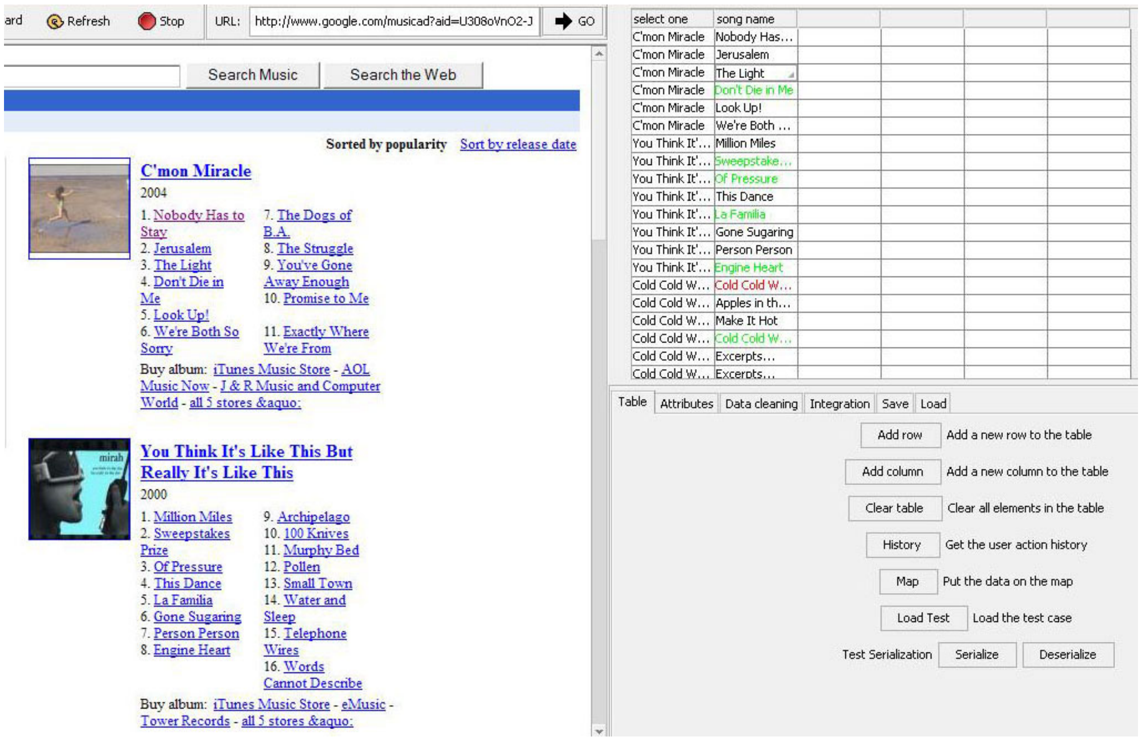


C'mon Miracle
2004
1. Nobody Has to Stay
2. Jerusalem
3. The Light
4. Don't Die in Me
5. Look Up!
6. We're Both So Sorry
7. The Dogs of B.A.
8. The Struggle
9. You've Gone Away Enough
10. Promise to Me
11. Exactly Where We're From
Buy album: iTunes Music Store - AOL Music Now - J & R Music and Computer World - all 5 stores &aquo;

You Think It's Like This But Really It's Like This
2000
1. Million Miles
2. Sweepstakes Prize
3. Of Pressure
4. This Dance
5. La Familia
6. Gone Sugaring
7. Person Person
8. Engine Heart
9. Archipelago
10. 100 Knives
11. Murphy Bed
12. Pollen
13. Small Town
14. Water and Sleep
15. Telephone Wires
16. Words Cannot Describe
Buy album: iTunes Music Store - eMusic - Tower Records - all 5 stores &aquo;

select one	song name
C'mon Miracle	Has to Stay
You Think It'...	Nobody Has to
Cold Cold W...	C: This is a list
To All We St...	Per I Morti R...
Joyride: Re...	The Light [T...
Advisory Co...	Cold Cold W...

Figure 3.10: Step 6: The user selects the cell (0,1) and chooses the option to specify that this column is a list.



C'mon Miracle
2004

1. Nobody Has to Stay
2. Jerusalem
3. The Light
4. Don't Die in Me
5. Look Up!
6. We're Both So Sorry
7. The Dogs of B.A.
8. The Struggle
9. You've Gone Away Enough
10. Promise to Me
11. Exactly Where We're From

Buy album: iTunes Music Store - AOL Music Now - J & R Music and Computer World - all 5 stores &aquo;

You Think It's Like This But Really It's Like This
2000

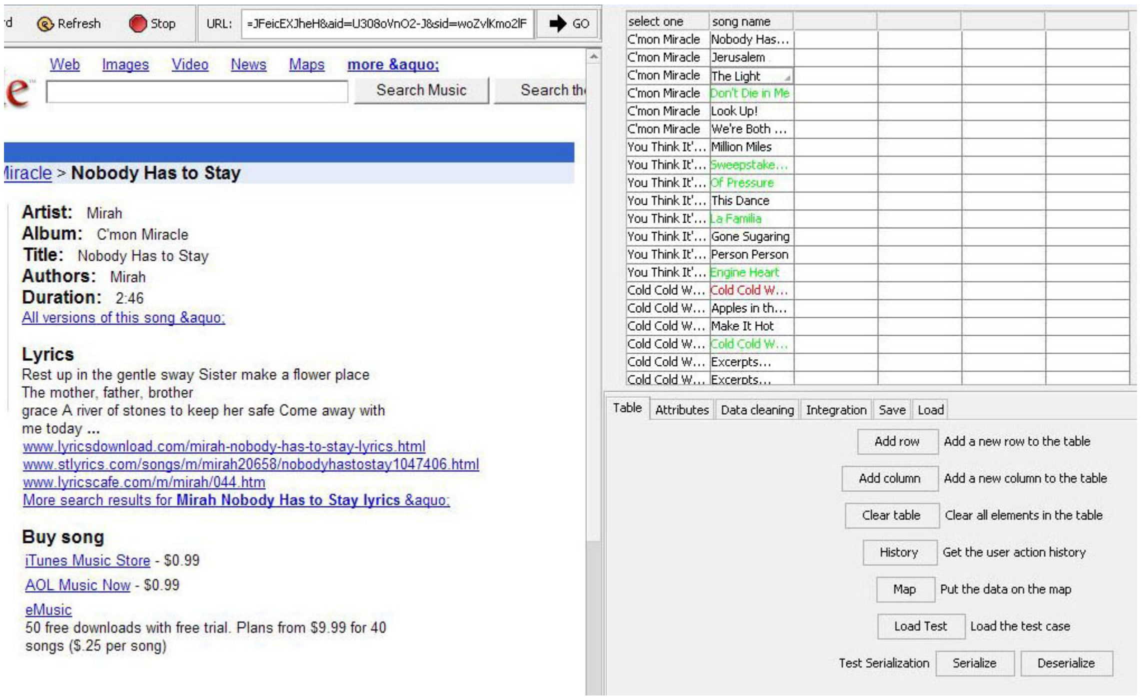
1. Million Miles
2. Sweepstakes Prize
3. Of Pressure
4. This Dance
5. La Familia
6. Gone Sugaring
7. Person Person
8. Engine Heart
9. Archipelago
10. 100 Knives
11. Murphy Bed
12. Pollen
13. Small Town
14. Water and Sleep
15. Telephone Wires
16. Words Cannot Describe

Buy album: iTunes Music Store - eMusic - Tower Records - all 5 stores &aquo;

select one	song name
C'mon Miracle	Nobody Has...
C'mon Miracle	Jerusalem
C'mon Miracle	The Light
C'mon Miracle	Don't Die in Me
C'mon Miracle	Look Up!
C'mon Miracle	We're Both ...
You Think It'...	Million Miles
You Think It'...	Sweepstake...
You Think It'...	Of Pressure
You Think It'...	This Dance
You Think It'...	La Familia
You Think It'...	Gone Sugaring
You Think It'...	Person Person
You Think It'...	Engine Heart
Cold Cold W...	Cold Cold W...
Cold Cold W...	Apples in th...
Cold Cold W...	Make It Hot
Cold Cold W...	Cold Cold W...
Cold Cold W...	Excerpts...
Cold Cold W...	Excerpts...

Add row	Add a new row to the table
Add column	Add a new column to the table
Clear table	Clear all elements in the table
History	Get the user action history
Map	Put the data on the map
Load Test	Load the test case
Test Serialization	Serialize / Deserialize

Figure 3.11: Step 7: Karma proceeds to extract every song name for every album and fills in the table.



iracle > **Nobody Has to Stay**

Artist: Mirah
Album: C'mon Miracle
Title: Nobody Has to Stay
Authors: Mirah
Duration: 2:46
All versions of this song &aquo;

Lyrics
Rest up in the gentle sway Sister make a flower place
The mother, father, brother
grace A river of stones to keep her safe Come away with
me today ...
www.lyricsdownload.com/mirah-nobody-has-to-stay-lyrics.html
www.stlyrics.com/songs/m/mirah20658/nobodyhastostay1047406.html
www.lyricscafe.com/m/mirah/044.htm
More search results for **Mirah Nobody Has to Stay lyrics** &aquo;

Buy song
iTunes Music Store - $0.99
AOL Music Now - $0.99
eMusic
50 free downloads with free trial. Plans from $9.99 for 40
songs ($.25 per song)

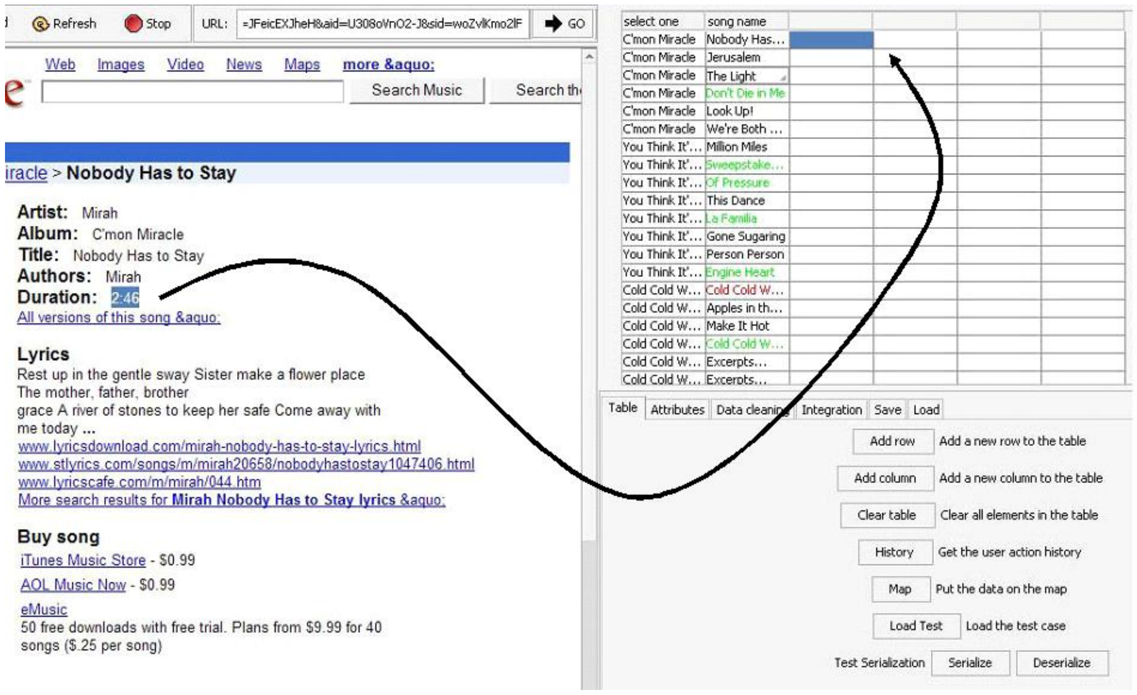
select one	song name			
C'mon Miracle	Nobody Has...			
C'mon Miracle	Jerusalem			
C'mon Miracle	The Light			
C'mon Miracle	Don't Die in Me			
C'mon Miracle	Look Up!			
C'mon Miracle	We're Both ...			
You Think It'...	Million Miles			
You Think It'...	Sweepstake...			
You Think It'...	Of Pressure			
You Think It'...	This Dance			
You Think It'...	La Familia			
You Think It'...	Gone Sugaring			
You Think It'...	Person Person			
You Think It'...	Engine Heart			
Cold Cold W...	Cold Cold W...			
Cold Cold W...	Apples in th...			
Cold Cold W...	Make It Hot			
Cold Cold W...	Cold Cold W...			
Cold Cold W...	Excerpts...			
Cold Cold W...	Excerpts...			

Add row	Add a new row to the table
Add column	Add a new column to the table
Clear table	Clear all elements in the table
History	Get the user action history
Map	Put the data on the map
Load Test	Load the test case
Test Serialization	Serialize / Deserialize

Figure 3.12: Step 8: The user now clicks on the link for the song *Nobody Has to Stay* to navigate to the song's detail page.

To extract every album name, we can relax the XPath by using the following expression: $\backslash html\backslash table*\backslash tr*\backslash td\backslash a\backslash b$. Instead of specifying the specific branch number, this XPath means: find all available permuted paths and return their values, if they exist. For example, $\backslash html\backslash table[2]\backslash tr[1]\backslash td\backslash a\backslash b$, one of the permuted XPaths, will return *You Think It's Like This But Really It's Like This*, while $\backslash html\backslash table[2\text{-}]\backslash tr[2]\backslash td\backslash a\backslash b$ will not return anything because there is no *a* node followed by a *b* node under that branch.

From sections 3.2.3 to 3.2.6, I will show how I exploit the DOM structure to perform data extraction and page navigation.



iracle > **Nobody Has to Stay**

Artist: Mirah
Album: C'mon Miracle
Title: Nobody Has to Stay
Authors: Mirah
Duration: 2:46
All versions of this song &aquo;

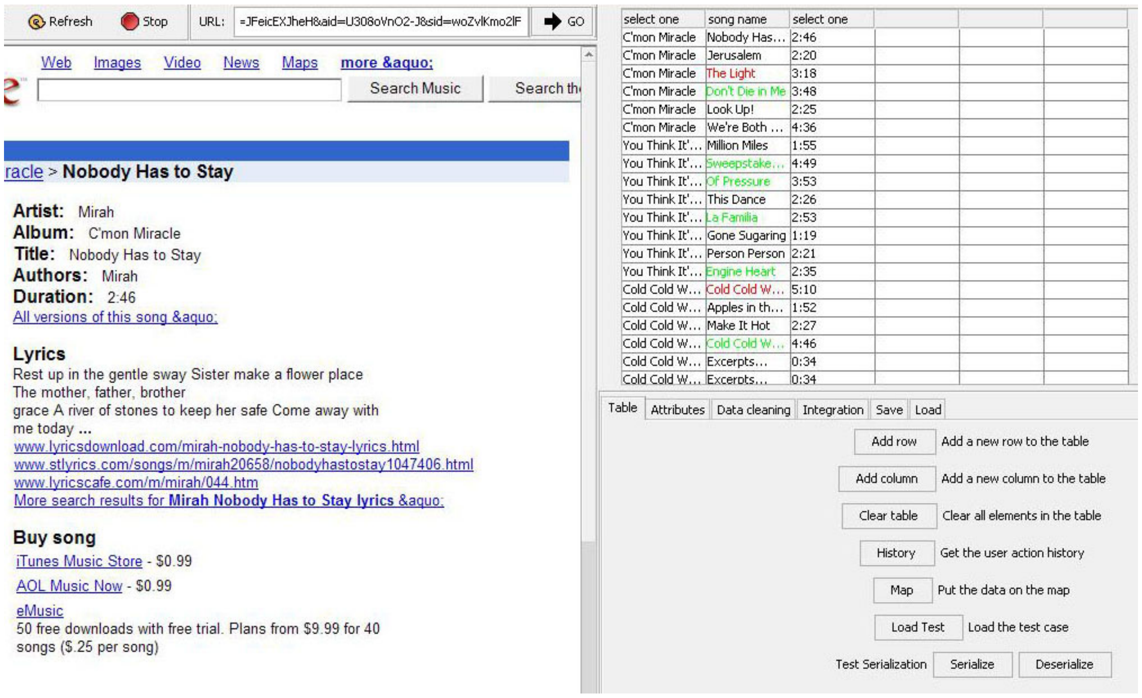
Lyrics
Rest up in the gentle sway Sister make a flower place
The mother, father, brother
grace A river of stones to keep her safe Come away with
me today **...**
www.lyricsdownload.com/mirah-nobody-has-to-stay-lyrics.html
www.stlyrics.com/songs/m/mirah20658/nobodyhastostay1047406.html
www.lyricscafe.com/m/mirah/044.htm
More search results for **Mirah Nobody Has to Stay lyrics** &aquo;

Buy song
iTunes Music Store - $0.99
AOL Music Now - $0.99
eMusic
50 free downloads with free trial. Plans from $9.99 for 40
songs ($.25 per song)

select one	song name				
C'mon Miracle	Nobody Has...				
C'mon Miracle	Jerusalem				
C'mon Miracle	The Light				
C'mon Miracle	Don't Die in Me				
C'mon Miracle	Look Up!				
C'mon Miracle	We're Both ...				
You Think It'...	Million Miles				
You Think It'...	Sweepstake...				
You Think It'...	Of Pressure				
You Think It'...	This Dance				
You Think It'...	La Familia				
You Think It'...	Gone Sugaring				
You Think It'...	Person Person				
You Think It'...	Engine Heart				
Cold Cold W...	Cold Cold W...				
Cold Cold W...	Apples in th...				
Cold Cold W...	Make It Hot				
Cold Cold W...	Cold Cold W...				
Cold Cold W...	Excerpts...				
Cold Cold W...	Excerpts...				

Add row	Add a new row to the table
Add column	Add a new column to the table
Clear table	Clear all elements in the table
History	Get the user action history
Map	Put the data on the map
Load Test	Load the test case

Figure 3.13: Step 9: The user extracts the duration of the song and drops it in the first row to signify that the newly extracted duration from the detail page is associated with the existing two data elements from the first page.



racle > **Nobody Has to Stay**

Artist: Mirah
Album: C'mon Miracle
Title: Nobody Has to Stay
Authors: Mirah
Duration: 2:46
All versions of this song &aquo;

Lyrics
Rest up in the gentle sway Sister make a flower place
The mother, father, brother
grace A river of stones to keep her safe Come away with
me today ...
www.lyricsdownload.com/mirah-nobody-has-to-stay-lyrics.html
www.stlyrics.com/songs/m/mirah20658/nobodyhastostay1047406.html
www.lyricscafe.com/m/mirah/044.htm
More search results for **Mirah Nobody Has to Stay lyrics** &aquo;

Buy song
iTunes Music Store - $0.99
AOL Music Now - $0.99
eMusic
50 free downloads with free trial. Plans from $9.99 for 40
songs ($.25 per song)

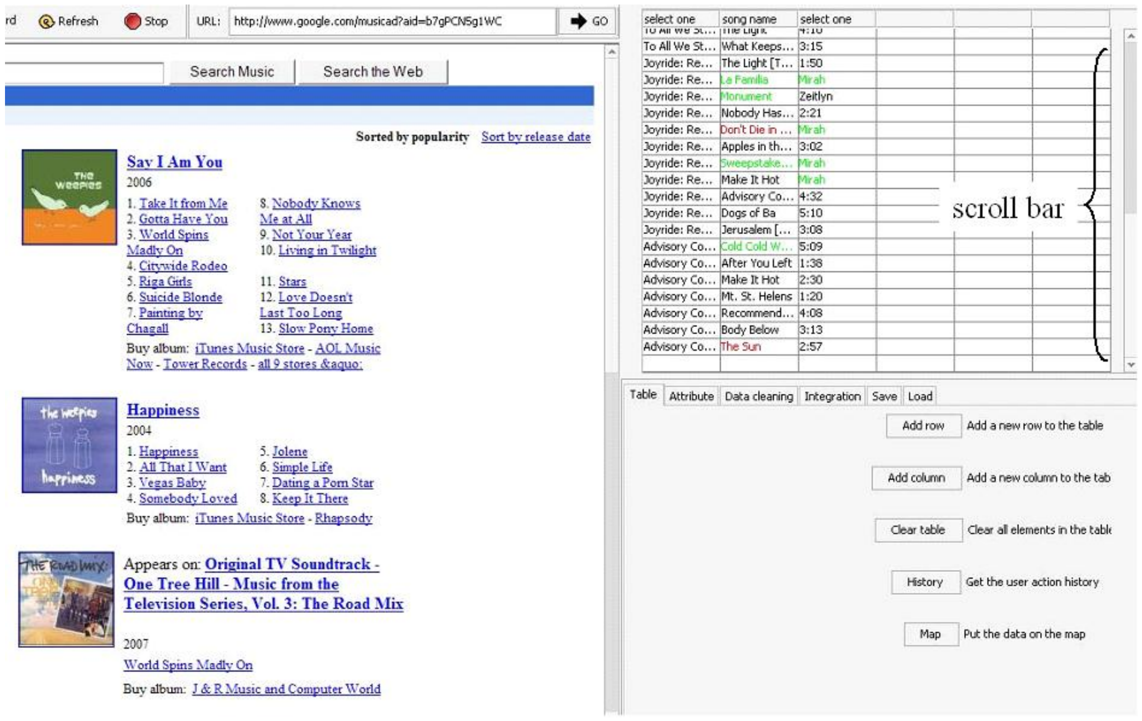
select one	song name	select one			
C'mon Miracle	Nobody Has...	2:46			
C'mon Miracle	Jerusalem	2:20			
C'mon Miracle	The Light	3:18			
C'mon Miracle	Don't Die in Me	3:48			
C'mon Miracle	Look Up!	2:25			
C'mon Miracle	We're Both ...	4:36			
You Think It'...	Million Miles	1:55			
You Think It'...	Sweepstake...	4:49			
You Think It'...	Of Pressure	3:53			
You Think It'...	This Dance	2:26			
You Think It'...	La Familia	2:53			
You Think It'...	Gone Sugaring	1:19			
You Think It'...	Person Person	2:21			
You Think It'...	Engine Heart	2:35			
Cold Cold W...	Cold Cold W...	5:10			
Cold Cold W...	Apples in th...	1:52			
Cold Cold W...	Make It Hot	2:27			
Cold Cold W...	Cold Cold W...	4:46			
Cold Cold W...	Excerpts...	0:34			
Cold Cold W...	Excerpts...	0:34			

Figure 3.14: Step 10: Karma proceeds to extract the duration for every song from each detail page and fills the table.

3.2.3 Using Markers to Enhance Extraction

The default DOM extraction works well when the list structure is flat. However, when there is a list within a list, more processing is needed in order to correctly extract the data. Our example is the case where there is a list within a list; song listings inside album listings. When the user selects *Nobody has to stay* and drops it onto the table, its corresponding XPath is *\html\table[1]\tr[2]\td\font\a[1]*. When we relax this XPath (*\html\table\*\tr*\td\font\a**), we end up extracting every song from every album. The problem is that we need contexts to associate each song to each specific album.

Figure 3.19 illustrates the situation. When the user puts *Nobody has to stay* on the same row as *C'mon miracle*, we know that the two data elements are associated. However,



Sorted by popularity Sort by release date

Say I Am You
2006
1. Take It from Me
2. Gotta Have You
3. World Spins Madly On
4. Citywide Rodeo
5. Riga Girls
6. Suicide Blonde
7. Painting by Chagall
8. Nobody Knows Me at All
9. Not Your Year
10. Living in Twilight
11. Stars
12. Love Doesn't Last Too Long
13. Slow Pony Home
Buy album: iTunes Music Store - AOL Music Now - Tower Records - all 9 stores &aquo;

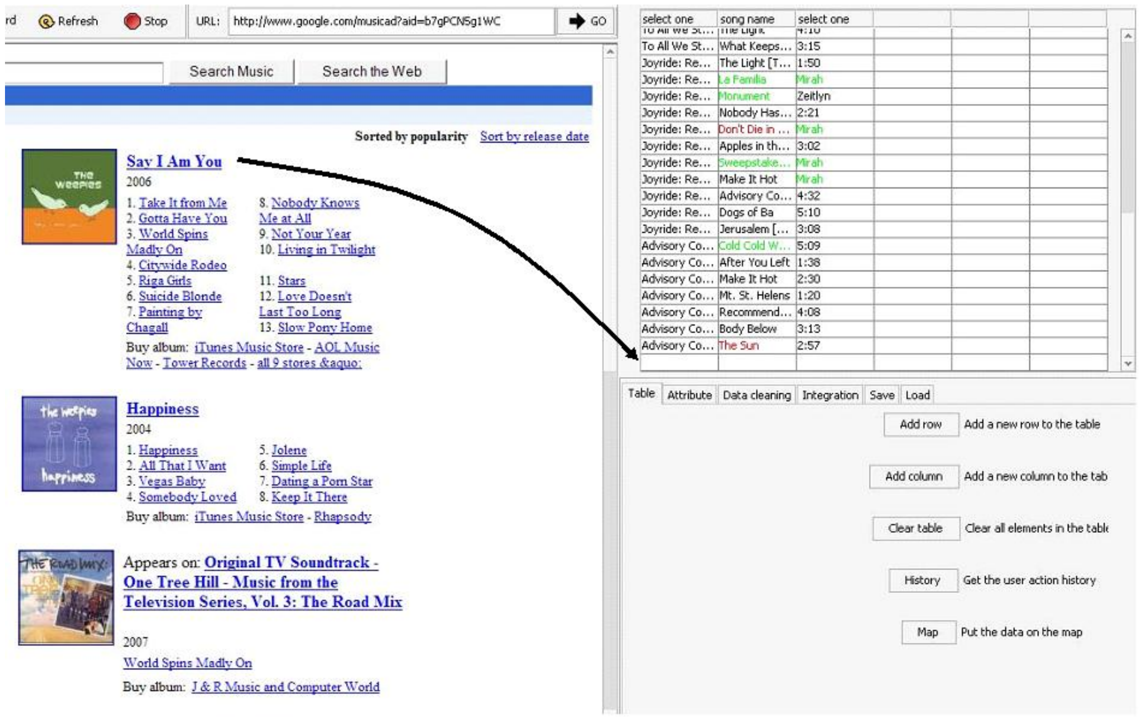
Happiness
2004
1. Happiness
2. All That I Want
3. Vegas Baby
4. Somebody Loved
5. Jolene
6. Simple Life
7. Dating a Porn Star
8. Keep It There
Buy album: iTunes Music Store - Rhapsody

Appears on: Original TV Soundtrack - One Tree Hill - Music from the Television Series, Vol. 3: The Road Mix
2007
World Spins Madly On
Buy album: J & R Music and Computer World

select one	song name	select one			
To All We St…	The Light	4:10			
To All We St…	What Keeps…	3:15			
Joyride: Re…	The Light [T…	1:50			
Joyride: Re…	La Familia	Mirah			
Joyride: Re…	Monument	Zeitlyn			
Joyride: Re…	Nobody Has…	2:21			
Joyride: Re…	Don't Die in …	Mirah			
Joyride: Re…	Apples in th…	3:02			
Joyride: Re…	Sweepstake…	Mirah			
Joyride: Re…	Make It Hot	Mirah			
Joyride: Re…	Advisory Co…	4:32			
Joyride: Re…	Dogs of Ba	5:10			
Joyride: Re…	Jerusalem […	3:08			
Advisory Co…	Cold Cold W…	5:09			
Advisory Co…	After You Left	1:38			
Advisory Co…	Make It Hot	2:30			
Advisory Co…	Mt. St. Helens	1:20			
Advisory Co…	Recommend…	4:08			
Advisory Co…	Body Below	3:13			
Advisory Co…	The Sun	2:57			

Add row	Add a new row to the table
Add column	Add a new column to the tab
Clear table	Clear all elements in the table
History	Get the user action history
Map	Put the data on the map

Figure 3.15: Step 11: The user now types in a new URL or browses to another artist's page. Note that the HTML structure of this new URL is similar to the original artist page.



Say I Am You
2006
1. Take It from Me
2. Gotta Have You
3. World Spins Madly On
4. Citywide Rodeo
5. Riga Girls
6. Suicide Blonde
7. Painting by Chagall
8. Nobody Knows Me at All
9. Not Your Year
10. Living in Twilight
11. Stars
12. Love Doesn't Last Too Long
13. Slow Pony Home
Buy album: iTunes Music Store - AOL Music Now - Tower Records - all 9 stores &aquo;

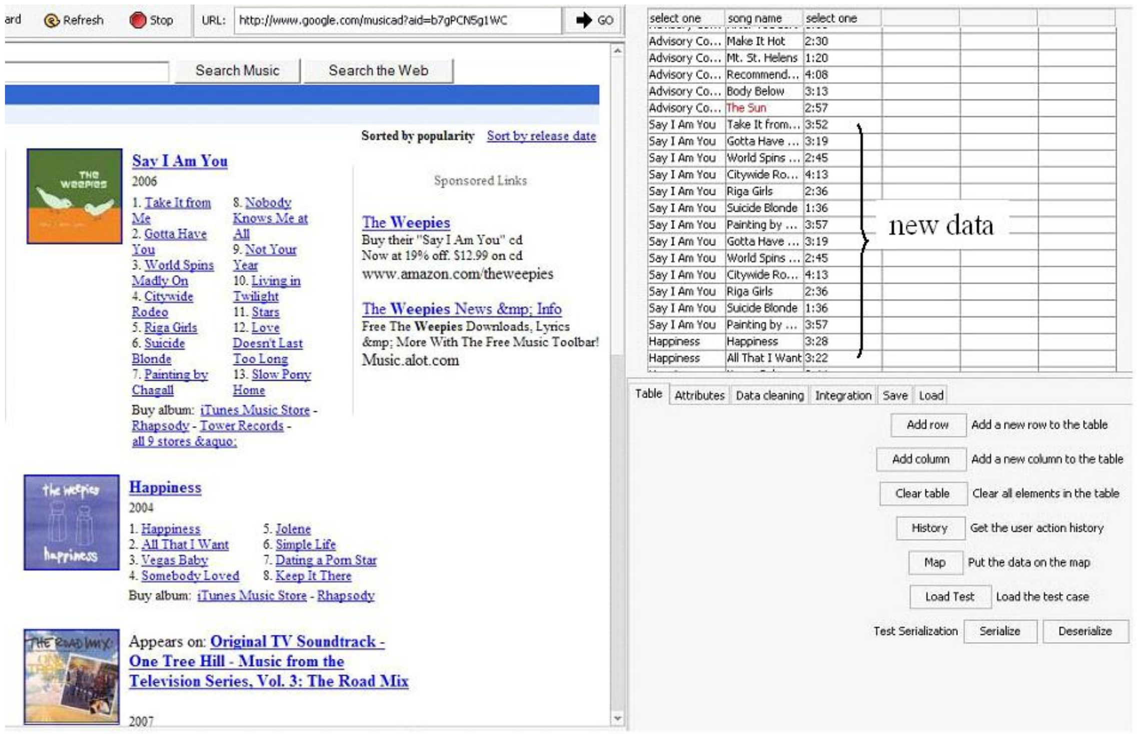
Happiness
2004
1. Happiness
2. All That I Want
3. Vegas Baby
4. Somebody Loved
5. Jolene
6. Simple Life
7. Dating a Porn Star
8. Keep It There
Buy album: iTunes Music Store - Rhapsody

Appears on: **Original TV Soundtrack - One Tree Hill - Music from the Television Series, Vol. 3: The Road Mix**
2007
World Spins Madly On
Buy album: J & R Music and Computer World

select one	song name	select one
To All We St…	The Light	4:10
To All We St…	What Keeps…	3:15
Joyride: Re…	The Light [T…	1:50
Joyride: Re…	La Familia	Mirah
Joyride: Re…	Monument	Zeitlyn
Joyride: Re…	Nobody Has…	2:21
Joyride: Re…	Don't Die in …	Mirah
Joyride: Re…	Apples in th…	3:02
Joyride: Re…	Sweepstake…	Mirah
Joyride: Re…	Make It Hot	Mirah
Joyride: Re…	Advisory Co…	4:32
Joyride: Re…	Dogs of Ba	5:10
Joyride: Re…	Jerusalem […	3:08
Advisory Co…	Cold Cold W…	5:09
Advisory Co…	After You Left	1:38
Advisory Co…	Make It Hot	2:30
Advisory Co…	Mt. St. Helens	1:20
Advisory Co…	Recommend…	4:08
Advisory Co…	Body Below	3:13
Advisory Co…	The Sun	2:57

Table
Attribute
Data cleaning
Integration
Save
Load
Add row
Add a new row to the table
Add column
Add a new column to the tab
Clear table
Clear all elements in the table
History
Get the user action history
Map
Put the data on the map

Figure 3.16: Step 12: The user extracts the album name of the new artist and drops it into a new row under the same column that contains album information from the first artist. This gives Karma a context that the new data is the same type as the data in column one, but from a different source.



Sorted by popularity Sort by release date

Say I Am You
2006
1. Take It from Me
2. Gotta Have You
3. World Spins Madly On
4. Citywide Rodeo
5. Riga Girls
6. Suicide Blonde
7. Painting by Chagall
8. Nobody Knows Me at All
9. Not Your Year
10. Living in Twilight
11. Stars
12. Love Doesn't Last Too Long
13. Slow Pony Home

Buy album: iTunes Music Store - Rhapsody - Tower Records - all 9 stores &aquo;

Sponsored Links

The Weepies
Buy their "Say I Am You" cd
Now at 19% off. $12.99 on cd
www.amazon.com/theweepies

The Weepies News &mp; Info
Free The **Weepies** Downloads, Lyrics &mp; More With The Free Music Toolbar!
Music.alot.com

Happiness
2004
1. Happiness
2. All That I Want
3. Vegas Baby
4. Somebody Loved
5. Jolene
6. Simple Life
7. Dating a Porn Star
8. Keep It There

Buy album: iTunes Music Store - Rhapsody

Appears on: **Original TV Soundtrack - One Tree Hill - Music from the Television Series, Vol. 3: The Road Mix**

2007

select one	song name	select one			
Advisory Co...	Make It Hot	2:30			
Advisory Co...	Mt. St. Helens	1:20			
Advisory Co...	Recommend...	4:08			
Advisory Co...	Body Below	3:13			
Advisory Co...	The Sun	2:57			
Say I Am You	Take It from...	3:52			
Say I Am You	Gotta Have ...	3:19			
Say I Am You	World Spins ...	2:45			
Say I Am You	Citywide Ro...	4:13			
Say I Am You	Riga Girls	2:36			
Say I Am You	Suicide Blonde	1:36			
Say I Am You	Painting by ...	3:57			
Say I Am You	Gotta Have ...	3:19			
Say I Am You	World Spins ...	2:45			
Say I Am You	Citywide Ro...	4:13			
Say I Am You	Riga Girls	2:36			
Say I Am You	Suicide Blonde	1:36			
Say I Am You	Painting by ...	3:57			
Happiness	Happiness	3:28			
Happiness	All That I Want	3:22			

Add row	Add a new row to the table
Add column	Add a new column to the table
Clear table	Clear all elements in the table
History	Get the user action history
Map	Put the data on the map
Load Test	Load the test case
Test Serialization	Serialize / Deserialize

Figure 3.17: Step 13: Karma proceeds to repeat all steps that the user did previously on the first source, including extracting each song name and its duration from each album. At this point, we have the table which contains the data from the two artists mentioned earlier in Figure 3.4.

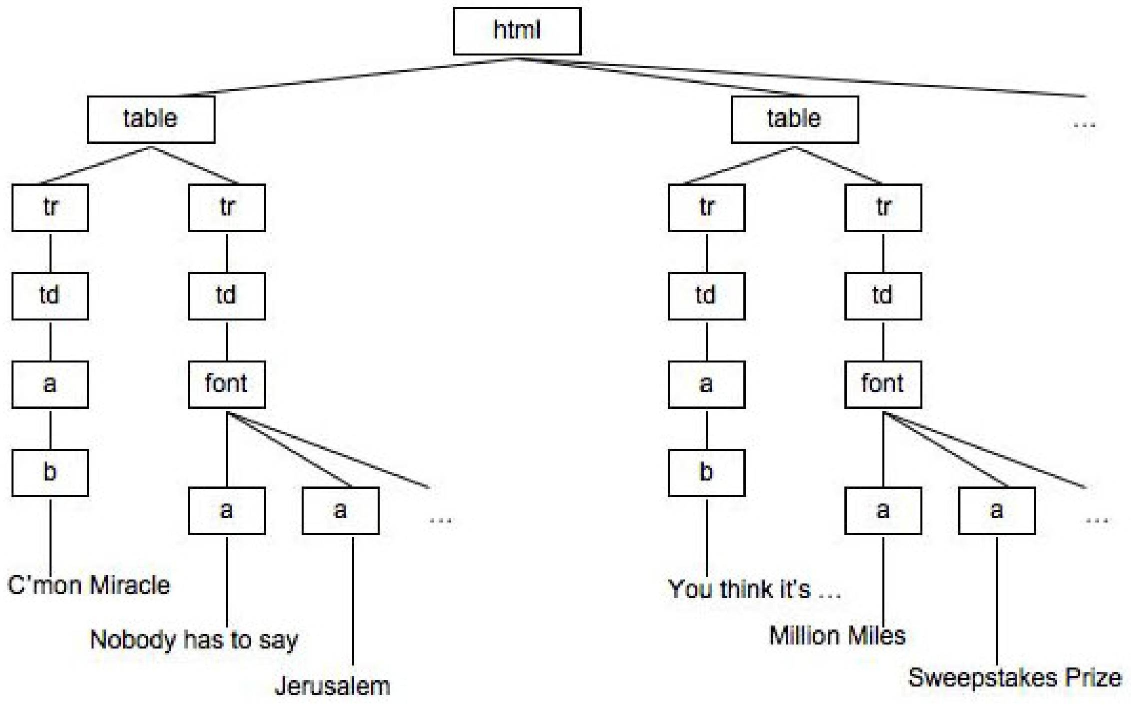


Figure 3.18: A *simplified* DOM representation of the album listing.

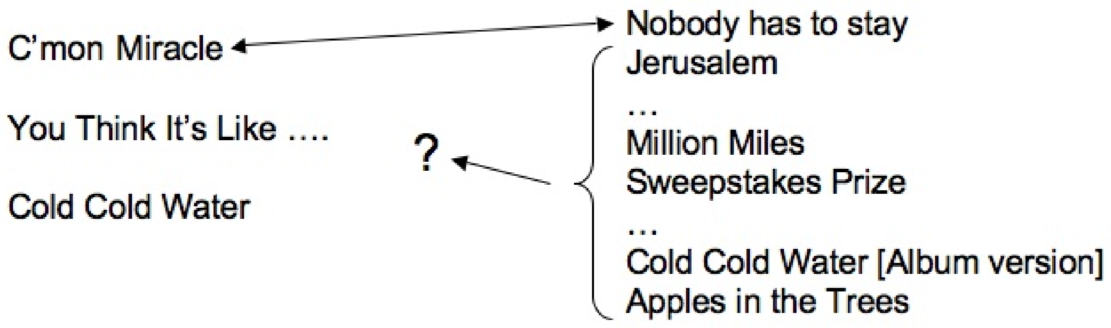


Figure 3.19: Using the default DOM approach to extract a list within a list will not work because there is no context to associate the data.

if we were to extract all the data, we would not know how to associate the rest of the songs with the extracted album.

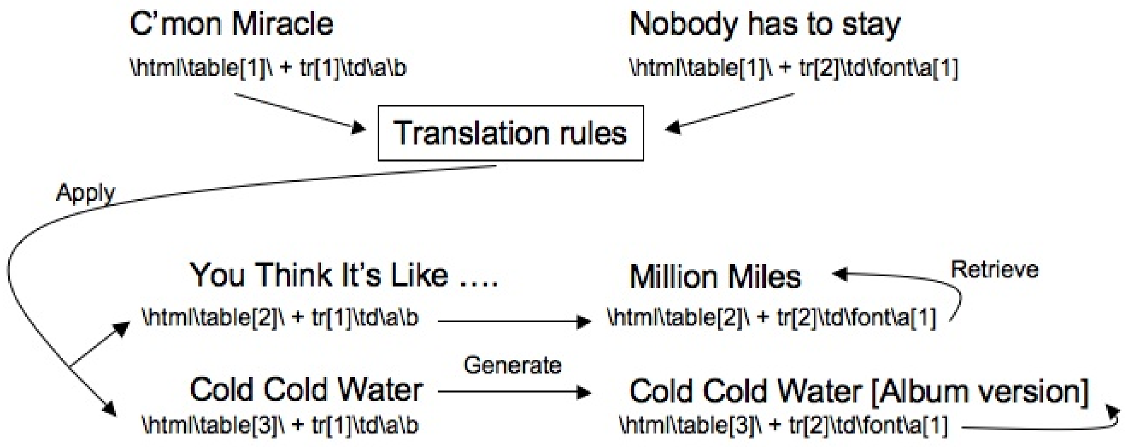

C'mon Miracle
\html\table[1]\ + tr[1]\td\a\b
Nobody has to stay
\html\table[1]\ + tr[2]\td\font\a[1]
Translation rules
Apply
You Think It's Like ....
\html\table[2]\ + tr[1]\td\a\b
Million Miles
\html\table[2]\ + tr[2]\td\font\a[1]
Retrieve
Generate
Cold Cold Water
\html\table[3]\ + tr[1]\td\a\b
Cold Cold Water [Album version]
\html\table[3]\ + tr[2]\td\font\a[1]

Figure 3.20: The workflow shows how to use context in the process of extracting a list within a list.

Figure 3.20 shows how to generate a rule between the following two XPaths:

XPath for *C'mon miracle*: *\html\table[1]\ + tr[1]\td\a\b*

XPath for *Nobody has to Stay*: *\html\table[1]\ + tr[2]\td\font\a[1]*

Notice that to generate the second XPath from the first XPath, we need a) the XPath portion that is common among the two XPath: *\html\table[1]* , and b) the XPath portion from the second XPath that differs from the first XPath: *tr[2]\td\font\a[1].*

The following rules are then applied to other XPaths in the album column to generate its corresponding song extraction.

1. Extract the common path *\html\table\** from the input XPath. Note that the XPath relaxation is needed to be able to find the common pattern from other XPaths.

2. Append the new common path with *tr[2]\td\font\a[1]* to locate a specific data element.

The general algorithm extracting new data rows after the first row has been extracted by using markers is shown below:

Precondition:

1. All the data extracted must come from the same page.
2. The user already extracted the data into the first column.

Let:

1. C_0 denotes the first column, where the data has been dragged and dropped into.
2. C_j denotes any arbitary column except the first column, where the user might drag and drop data into.
3. R_{ij} denotes a cell (i,j) in the table where i corresponds to the row and j corresponds to the column.
4. T_{xy} denotes a translation rule between C_x and C_y where given an XPath of R_{xj}, T_{xy} returns an XPath of R_{yj}.

Assuming the user drag and drops a new data element into a new empty column at cell R_{ij}, Karma uses the following algorithm to help fill every cell in C_j:

1. Create T_{xy} by using the following two XPaths as inputs (R_{i0}, R_{ij}).
2. Apply T_{xy} to each cell $R_{0..n,0}$ in C_0 to generate its corresponding XPath for each cell $R_{0..n,j}$ in C_j.

3. For each XPath generated with respect to a cell in $R_{x,j}$ (where x can ranges from 0 to n), Karma executes that XPath to retrieve the data from the source page and fills the data in the cell.

To compute T_{xy} given R_{i0} and R_{ij}, the following algorithm is used:

1. Compute the longest common substring between the XPath of R_{i0} and the XPath of R_{ij}. Let us designate this terms as C_s.

2. The XPath of R_{ij} can be defined as $C_s + L_s$, where L_s is the substring that does not belong to the longest common substring.

3. Since C_s is also an XPath, we relax C_s and replace the *[number]* portion of that XPath with *. For example, \tr[2]\td[1] will become \tr*\td*. Let the relax string be defined as F_s

Once T_{xy} is computed, when given an arbitary XPath of $R_{x,i}$, we can compute an XPath for $R_{y,i}$ by:

1. First, try to locate a substring in the XPath of $R_{x,i}$. Let's call this string M_s. When M_s is relaxed, it should be equal to F_s.

2. Once M_s is found, the XPath for $R_{y,i}$ is basically $M_s + L_s$.

3.2.4 Horizontal Expansion

As shown in the case study walkthrough section, sometimes the data that we need spans multiple pages from the same web sites. In general, the first page might contain a listing of data with links to a specific detail page for each data element. When the user browses to the detail page of a particular song and extracts the song's duration, we want to be able to automatically extract the duration of other songs and fill them in the table.

When the user drag and drops the song's duration into the first row, she indirectly specifies the context that the data *2:46* is related to two data elements: *C'mon Miracle* and *Nobody Has to Stay*. However, since the user never specifies which one, we need to determine which element we should use as a marker.

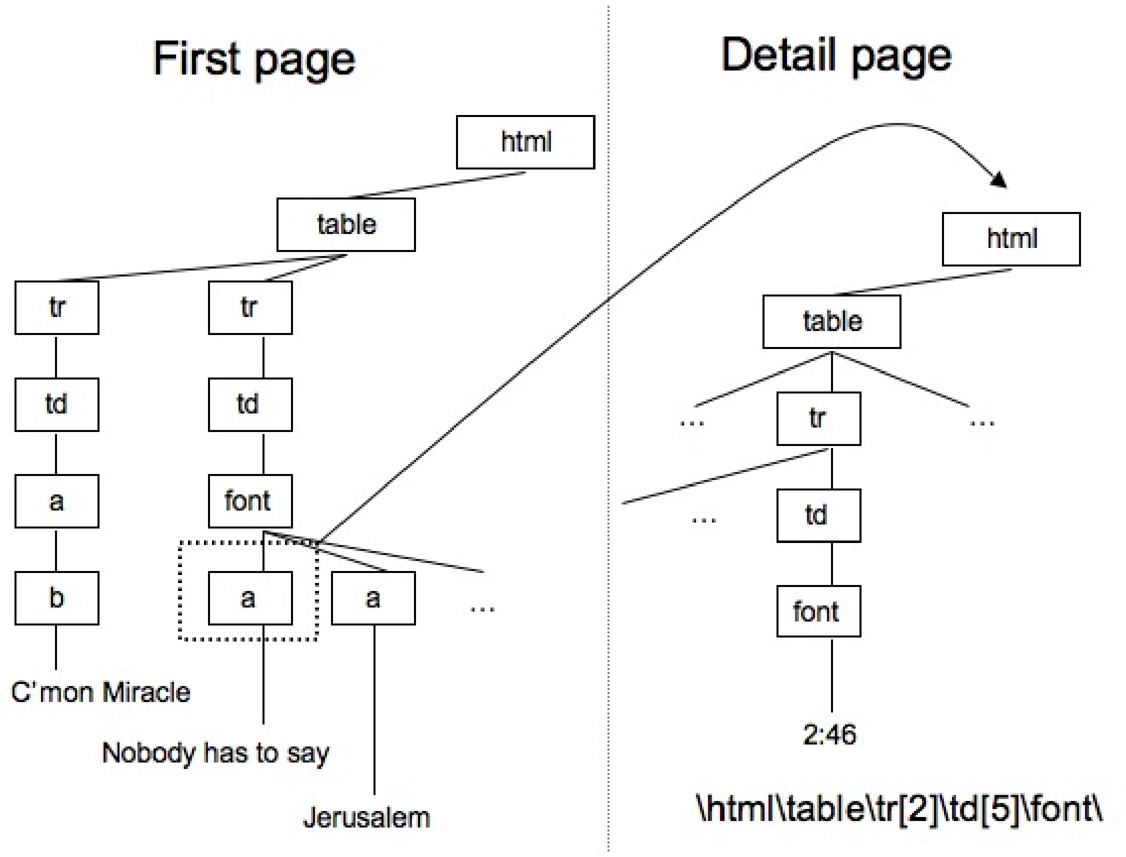
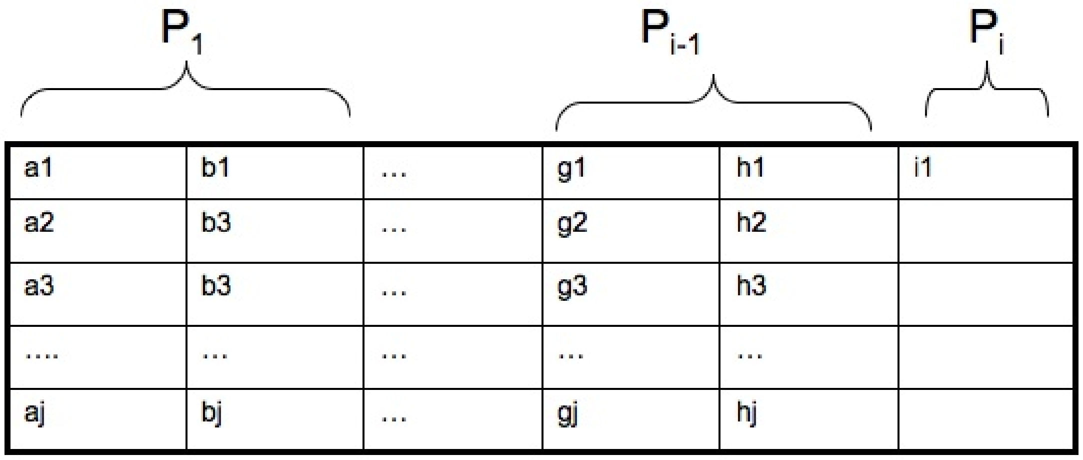


Figure 3.21: The relationship between the first page and one of its detail page. Figuring out the correct marker is the key to automatically extracting data from detail pages.

Figure 3.21 shows the relationship between the first page and the detail page. When the user clicks a link to go to the detail page, it is possible to identify the XPath of the node that contains the URL information that the user just clicked. In our example, this

XPath is: \html\\a[@href=URL of the detail page]. This XPath will return a node in the first page that contains an *a* node with its *href* attribute equals to the URL of the detail page that the user picks. In our case, this node is: \html\table[1]\\ + *tr[2]\td\font\a[1]*.

Once we have this node, we need to figure out the closest node between this node and existing nodes from the same row that comes from the first page. In this case, we can see that XPath for *Nobody Has to Stay* is the same as the XPath for the node that contains the correct *href* attribute. As a result, the *Nobody has to stay* node is chosen, along with the rest of the nodes in the same column, as a marker for the horizontal expansion. The translation rule that we want to create is: given the horizontal marker as an input, return its corresponding node that contains the *href* information as an output. This requires two steps: a) generate an XPath for the node that contains the *href* information, which is similar to the translation rule shown earlier; and b) extract the URL of the detail page from that node. This can be done using Xquery.



a1	b1	...	g1	h1	i1
a2	b3	...	g2	h2	
a3	b3	...	g3	h3	
....	...	...	...	...	
aj	bj	...	gj	hj	

Figure 3.22: Abstract representation of the horizontal expansion case.

After we have the horizontal translation rule, we can apply this rule to each element under the horizontal marker column to gather all the detail pages for other rows. Then, we can apply the XPath that extracts the duration (*\html\table\tr[2]\td[5]\font*) shown in Figure 3.21 to each detail page to extract the time data for each song on the table.

Figure 3.22 shows the horizontal expansion case in a more abstract way.

Let:

1. $[a-z]_{1..i}$ represents the data element extracted from web pages.
2. P_i represents a web page at depth i. For example, P_1 is the top page, P_2 is the child page under P_1, and so on.
3. URL_{P_i} represents the URL of the page P_i.
4. C_i represents the column of the table that contains the data element $i_{1..j}$.
5. X_{a_i} represents the XPath of the data element a_i.

Given that the user extracts the data element i_1 from the page P_i, we can fill the column C_i using the following algorithm:

1. Locate the data elements on row 1 that belong to the page P_{i-1}. In this case, it is the set {g1, h1}.
2. Generate an Xquery that locates the node in P_{i-1} that contains the *href* attribute that is equal to URL_{P_i}. Let us assume that node y is returned.
3. Designate a marker by comparing the distance between X_y and the XPath of each element of the set {g1, h1}. The distance can be computed by counting the length

of the common path between X_y and another node. Let's assume that the X_{h_1} is the closest and is designated as the horizontal expansion marker.

4. Generate the horizontal translation rule between X_{h_1} and X_y so that given X_{h_1} as an input, we can generate X_y as an output.

5. Apply the horizontal translation rule to every element in C_h (i.e., $X_{h_2}...X_{h_j}$)to locate the detail page for each row.

6. For each detail page generated, apply X_{i_1} to the detail page's DOM tree to extract the data and fill it in the table.

Using this approach, it is possible for the user to keep navigating into multiple detailed page levels, extracting data, and integrating data between these pages at the same time.

3.2.5 Vertical Expansion

After extracting album listings, song listings, and their corresponding duration for the first artist, Karma can reuse generated markers, translation rules, and horizontal translation rules to extract pages with the same structure. Specifically, these marker positions and rules constitute a wrapper that is capable of extracting data from a similar page. In our case study example, after the user finishes extracting information about the artist Mirah, she also wants to extract the same kind of information from another artist. By drag and dropping one of the album names into a new row under the first column, the user indirectly specifies the context that the new data element is the same kind as other data elements in the same column. However, since the new data element is from a different URL compared to other data elements in the same column, Karma deduces that this is a vertical expansion operation where the rules and steps generated for the original pages should be also be applied to this new URL.

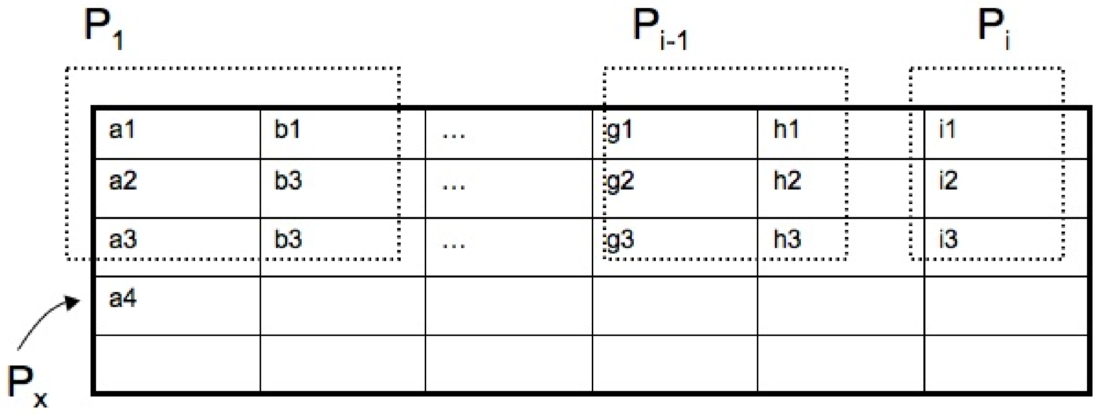


Figure 3.23: Abstract representation of the vertical expansion case.

Figure 3.23 shows the vertical expansion case in a more abstract way. In this scenario, the first three rows correspond to the data extraction that originally starts from P_1, while the new element a_4 is extracted from the different start page P_x.

Preconditions:

1. The table is not empty.

2. a_4 is the only element in its row.

3. URL of a_4 is different from URL of other elements under the same column

If all the preconditions satisfy, then following algorithm, using the same terminology defined earlier, for vertical expansion is shown below:

1. Relax X_{a_4} to extract the list that X_{a_4} is from.

2. Apply the translation rule generated between a_1 and b_1 to each of the new elements under C_a. These elements include X_{a_4} and other data elements generated by the relaxation step.

3. Locate the column that is used as the horizontal expansion marker and apply the horizontal expansion algorithm discussed earlier to the new set of data.

4. Continue to apply the horizontal expansion algorithm through multiple detailed pages ($P_{2...i}$) until done.

At the end of this algorithm, Karma will repeat all the extract steps done in the first three rows on the page where X_{a_4} is from similar to invoking a predefined wrapper on a new input page.

3.2.6 Form Extraction

Many web sites contain HTML forms where a user needs to type in a query in order to get a result. Figure 3.24 shows a shopping web site that has a form input. Karma captures this form automatically as the user interacts with it and allows the user to invoke the form from within the table area. Figure 3.24 to Figure 3.30 shows an interaction scenario.

Initially, the user enters the search term *english* in the search box as shown in Figure 3.24 and presses *go*. The form then redirects the embedded browser to a new page. Figure 3.25 shows the partial snapshot of the result page. At the same time, Karma captures the form parameters and fills them into the table as shown in Figure 3.26.

From this point on, the user continues to extract the data normally by highlight-drag-drop a data item that belongs to a list and Karma will fill in the rest of the list

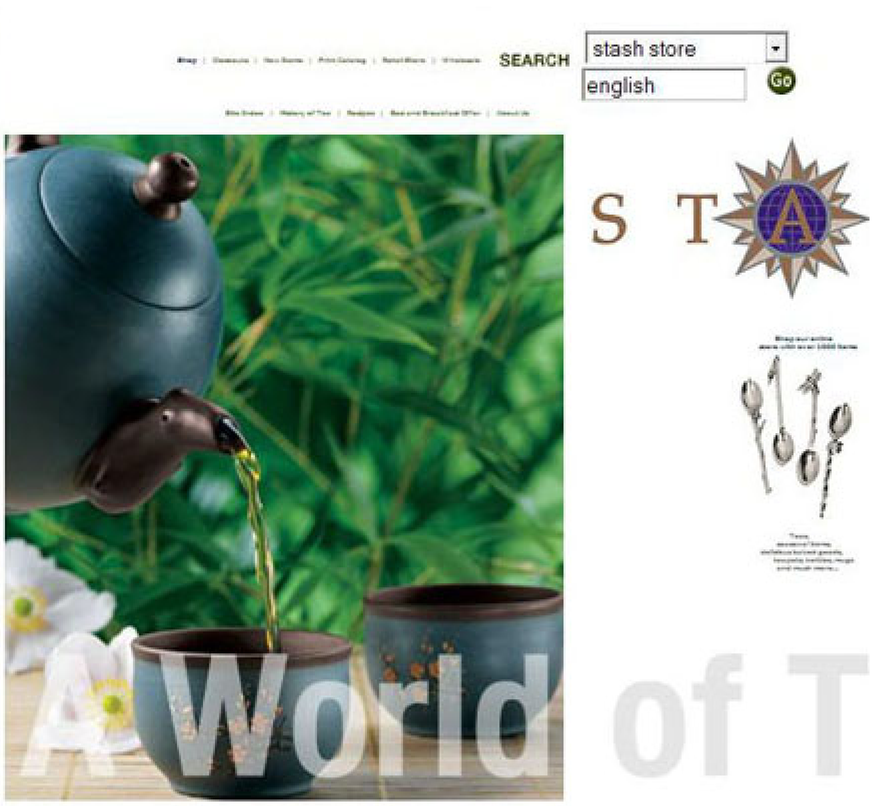
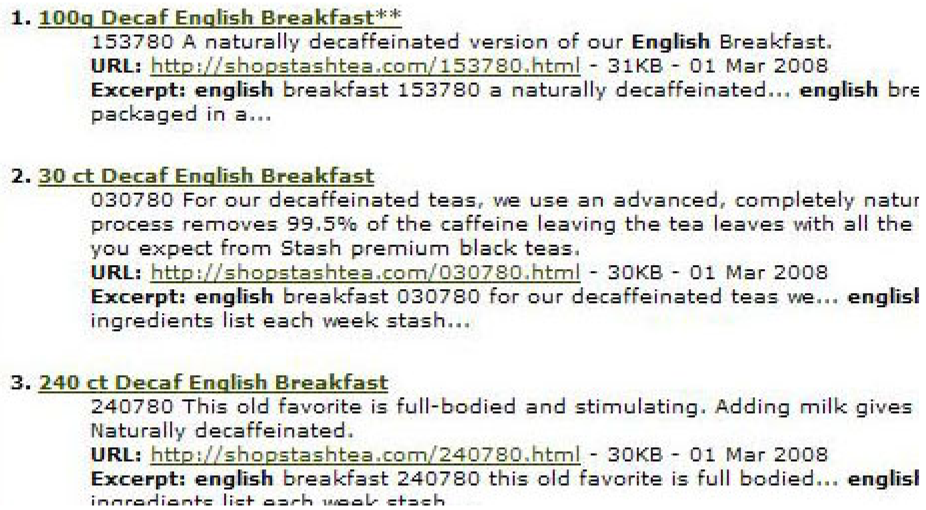


Figure 3.24: An example website (www.stashtea.com) that has an input form.

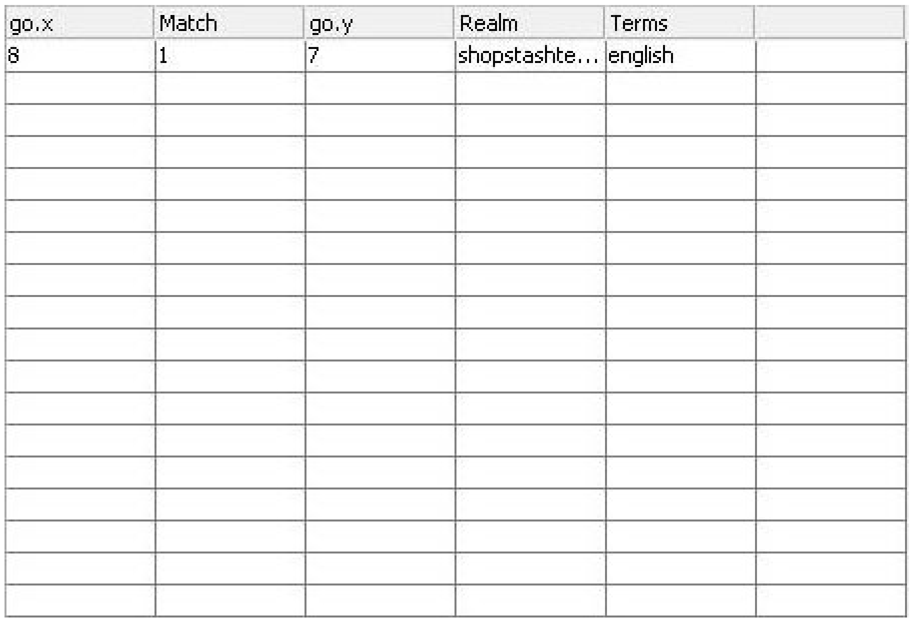


1. 100g Decaf English Breakfast**
153780 A naturally decaffeinated version of our **English** Breakfast.
URL: http://shopstashtea.com/153780.html - 31KB - 01 Mar 2008
Excerpt: english breakfast 153780 a naturally decaffeinated... **english** bre
packaged in a...

2. 30 ct Decaf English Breakfast
030780 For our decaffeinated teas, we use an advanced, completely natur
process removes 99.5% of the caffeine leaving the tea leaves with all the
you expect from Stash premium black teas.
URL: http://shopstashtea.com/030780.html - 30KB - 01 Mar 2008
Excerpt: english breakfast 030780 for our decaffeinated teas we... **englisl**
ingredients list each week stash...

3. 240 ct Decaf English Breakfast
240780 This old favorite is full-bodied and stimulating. Adding milk gives
Naturally decaffeinated.
URL: http://shopstashtea.com/240780.html - 30KB - 01 Mar 2008
Excerpt: english breakfast 240780 this old favorite is full bodied... **englisl**
ingredients list each week stash...

Figure 3.25: The result page when a user enters *english* as a search term.

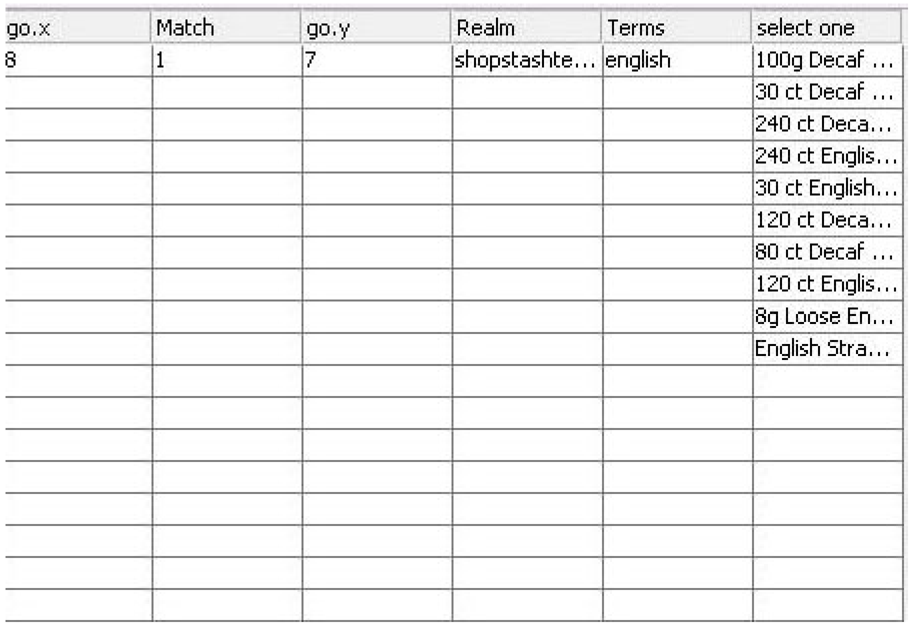


go.x	Match	go.y	Realm	Terms	
8	1	7	shopstashte...	english	

Figure 3.26: Karma captures every form parameter and automatically fills them in the table.

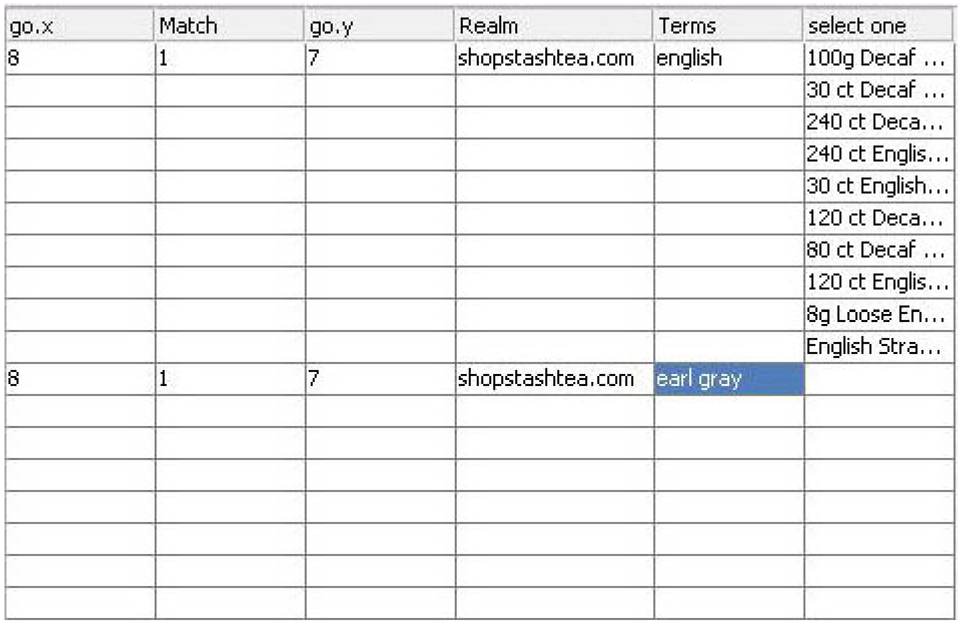
automatically as shown in Figure 3.27. After the user is done with the result page generated from the query term *english*, the user can invoke the form from within the table by entering a value for each form parameter in a new empty row. Note that for other form parameters besides the query parameter, the user can enter previous values. Figure 3.28 shows the user enters the new set of parameters to search for *earl gray*.

When the user enters a new set of parameters, Karma will invoke the form and show the new result page (Figure 3.29) to the user. The user can then proceed to extract the data from the new result page onto the table as shown in Figure 3.30. If the user had extracted more than one column of data or had done any horizontal expansion from the first query result, those processes would be repeated automatically after the user drops the first data item from the second query (i.e., *30 ct Double....*) onto the table.



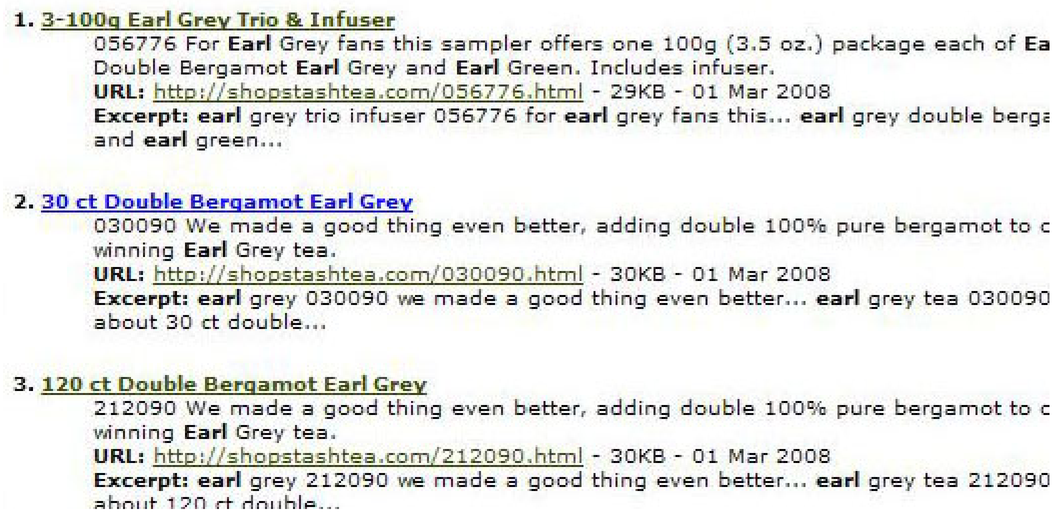
go.x	Match	go.y	Realm	Terms	select one
8	1	7	shopstashte...	english	100g Decaf ...
					30 ct Decaf ...
					240 ct Deca...
					240 ct Englis...
					30 ct English...
					120 ct Deca...
					80 ct Decaf ...
					120 ct Englis...
					8g Loose En...
					English Stra...

Figure 3.27: The user can proceed to extract the data the same way as shown in the previous example.



go.x	Match	go.y	Realm	Terms	select one
8	1	7	shopstashtea.com	english	100g Decaf ...
					30 ct Decaf ...
					240 ct Deca...
					240 ct Englis...
					30 ct English...
					120 ct Deca...
					80 ct Decaf ...
					120 ct Englis...
					8g Loose En...
					English Stra...
8	1	7	shopstashtea.com	earl gray	

Figure 3.28: The user can also invoke the form from within the table by typing in new parameters in an empty row. In this case, the user specify the value for the parameter *term* to be *earl gray*

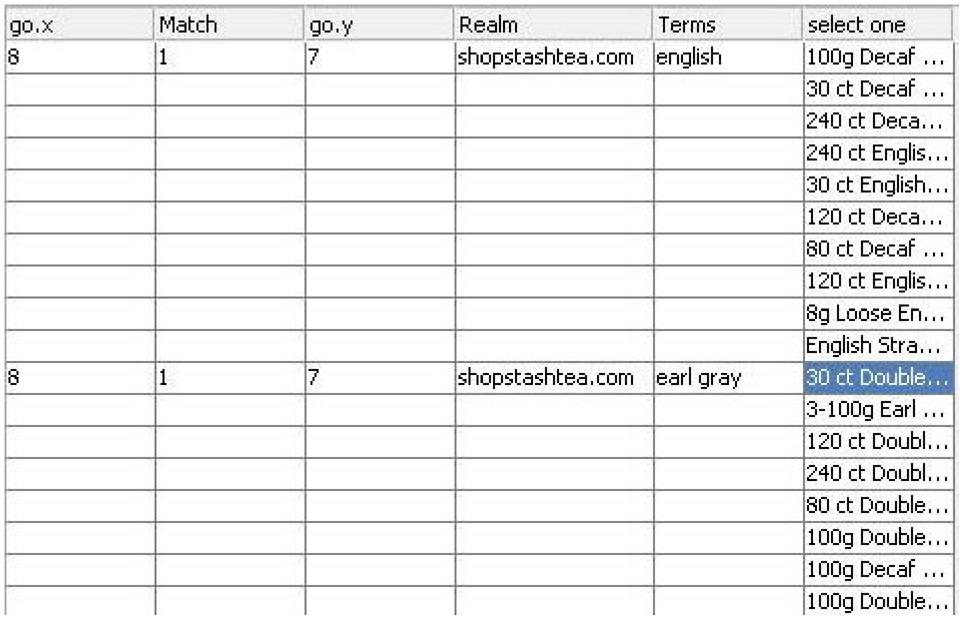


1. 3-100g Earl Grey Trio & Infuser
056776 For **Earl** Grey fans this sampler offers one 100g (3.5 oz.) package each of **Ea**
Double Bergamot **Earl** Grey and **Earl** Green. Includes infuser.
URL: http://shopstashtea.com/056776.html - 29KB - 01 Mar 2008
Excerpt: earl grey trio infuser 056776 for **earl** grey fans this... **earl** grey double berga
and **earl** green...

2. 30 ct Double Bergamot Earl Grey
030090 We made a good thing even better, adding double 100% pure bergamot to c
winning **Earl** Grey tea.
URL: http://shopstashtea.com/030090.html - 30KB - 01 Mar 2008
Excerpt: earl grey 030090 we made a good thing even better... **earl** grey tea 030090
about 30 ct double...

3. 120 ct Double Bergamot Earl Grey
212090 We made a good thing even better, adding double 100% pure bergamot to c
winning **Earl** Grey tea.
URL: http://shopstashtea.com/212090.html - 30KB - 01 Mar 2008
Excerpt: earl grey 212090 we made a good thing even better... **earl** grey tea 212090
about 120 ct double...

Figure 3.29: The embedded browser refreshes to reflect the new result page.



go.x	Match	go.y	Realm	Terms	select one
8	1	7	shopstashtea.com	english	100g Decaf ...
					30 ct Decaf ...
					240 ct Deca...
					240 ct Englis...
					30 ct English...
					120 ct Deca...
					80 ct Decaf ...
					120 ct Englis...
					8g Loose En...
					English Stra...
8	1	7	shopstashtea.com	earl gray	30 ct Double...
					3-100g Earl ...
					120 ct Doubl...
					240 ct Doubl...
					80 ct Double...
					100g Double...
					100g Decaf ...
					100g Double...

Figure 3.30: The user then extracts the data by dragging only one data from the list. If the user has previously extracted more than the listing, the horizontal and vertical expansion process will start up and automatically repeat all the steps done.

Karma captures the form when the user submits the query by looking at the result URL. For example, the result URL from the search term *english* is:

http://stashtea.com/cgi-bin/search/search.pl?Realm=shopstashtea.com&Terms=english&Match=1&go.x=0&go.y=0

This URL can be broken down into two portions: the query portion and the parameter portion. The query portion starts from the beginning of the URL to the symbol ?: *http://stashtea.com/cgi-bin/search/search.pl?*; the rest belongs to the parameter portion. The parameter portion has the following syntax: *param1=value1&param2=value2.....&paramN=valueN*. When the user fills in the new set of values, Karma then constructs the new URL based on those values and retrieves the new result page.

3.3 Discussion

The data retrieval approach shown in this chapter fits the Karma framework by primarily focusing on the data, not the operations. Instead of writing complex extraction and navigation rules, the user retrieves data by moving data from web pages in the embedded browser to the table. The user is able to extract data, including an HTML form, from multiple levels of web pages and to combine them by indirectly giving positional contexts. The data retrieval process is based on enhancing the DOM and exploiting positional context in the table.

Each table has two obvious but often neglected constraints, which define the positional context technique. The first constraint is the horizontal constraint. When a user puts a new data element in a new column on a particular row, the user indirectly associates the new element with other elements in the same row. It means the new element has a different semantic because it is in a different column, but all elements in that row belong to the same tuple. The second constraint is the vertical constraint. When the user puts a new data element in a particular column, the user indirectly associates the new element with other elements in the same column. It means the new element has the same semantic as other elements in that column.

The horizontal constraint helps Karma identify markers in order to connect and combine data from multiple page levels, while the vertical constraint signals Karma that the user wants to invoke the rules and steps learned previously on a new, but similar source. The idea of positional context will be revisited in chapter 6.

As soon as the user puts the data into the table, the new data elements are pipelined into the source modeling problem phrase. For example, the attribute name of the second column in Figure 3.9 is automatically set to *song name*. Also, data elements with lighter color shown in Figure 3.11 are the result from the data cleaning phrase. More details will be elaborated in the next two chapters.

Chapter 4

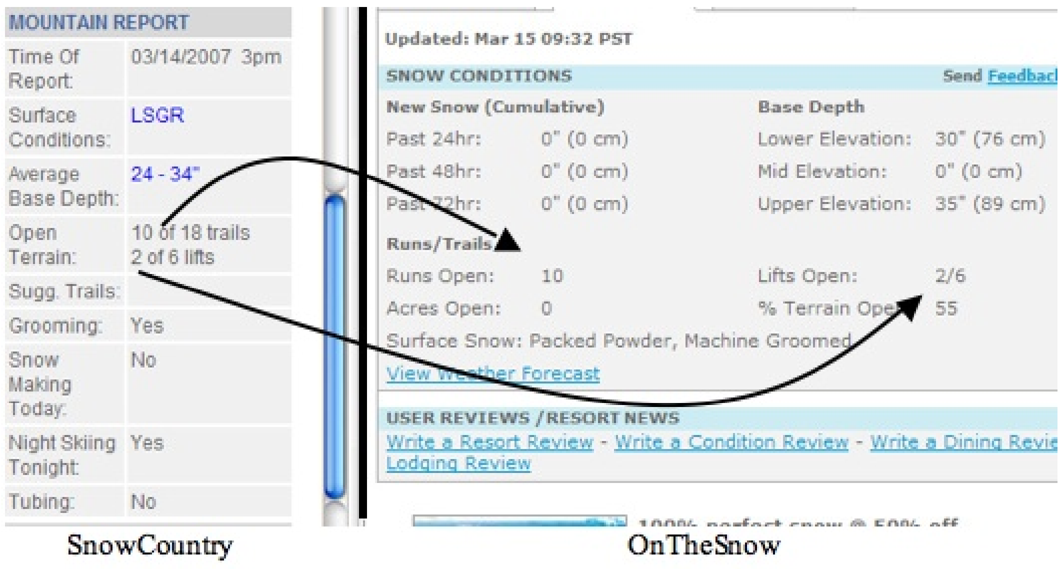
Source Modeling

This chapter is divided into three sections. The first section defines what source modeling is and why it is important. The second section first shows how a user would interact with Karma to solve the source modeling problem and then describes the technical details behind those interactions. The third section elaborates on how my approach to the source modeling problem fits the overall Karma framework.

4.1 Problem Definition

Source modeling is the process of mapping a new source to existing sources in terms of attributes. Knowing how each new source maps to existing sources allows Karma to identify potential sources to be used in data cleaning (as shown in Figure 2.15) and data integration.

Figure 4.1 shows two HTML pages from two different data sources. Each page provides a detailed snow report of the Edelweiss resort. Assuming that **OnTheSnow** is the source that already exists in the database, the source modeling task is to map attributes from



MOUNTAIN REPORT

Time Of Report:	03/14/2007 3pm
Surface Conditions:	LSGR
Average Base Depth:	24 - 34"
Open Terrain:	10 of 18 trails 2 of 6 lifts
Sugg. Trails:	
Grooming:	Yes
Snow Making Today:	No
Night Skiing Tonight:	Yes
Tubing:	No

Updated: Mar 15 09:32 PST

SNOW CONDITIONS Send Feedbac

New Snow (Cumulative)		Base Depth	
Past 24hr:	0" (0 cm)	Lower Elevation:	30" (76 cm)
Past 48hr:	0" (0 cm)	Mid Elevation:	0" (0 cm)
Past 72hr:	0" (0 cm)	Upper Elevation:	35" (89 cm)
Runs/Trails			
Runs Open:	10	Lifts Open:	2/6
Acres Open:	0	% Terrain Open:	55

Surface Snow: Packed Powder, Machine Groomed

View Weather Forecast

USER REVIEWS / RESORT NEWS

Write a Resort Review - Write a Condition Review - Write a Dining Revie Lodging Review

Figure 4.1: An example showing the difficulty of solving the source modeling problem.

SnowCountry to attributes from **OnTheSnow**. Note that it is not always possible to map all the attributes from a new data source to existing sources.

In addition, trying to identify the mapping can prove to be difficult. For example, the attribute *Time Of Report* from SnowCountry maps to the attribute *Updated* from OnTheSnow. However, trying to identify this mapping will require:

- Some way to identify that the value under an attribute is a date/time
- Data cleaning to translate those values into the same format
- A way to deal with dynamic data; the snow data reported is taken from different times.

Another example is how *Open Terrain* from **SnowCountry** is mapped to two different attributes (*Runs Open* and *Lifts Open*) from OnTheSnow. In this case, one attribute from one source can be mapped to multiple attributes from another source. Checking for possible mapping candidates will require the creation of multiple permutations of potential candidate sets. This problem is often referred to as *n:m* mapping, in contrast to the *1:1* mapping case where an attribute from one source can be mapped to only a single attribute from another source.

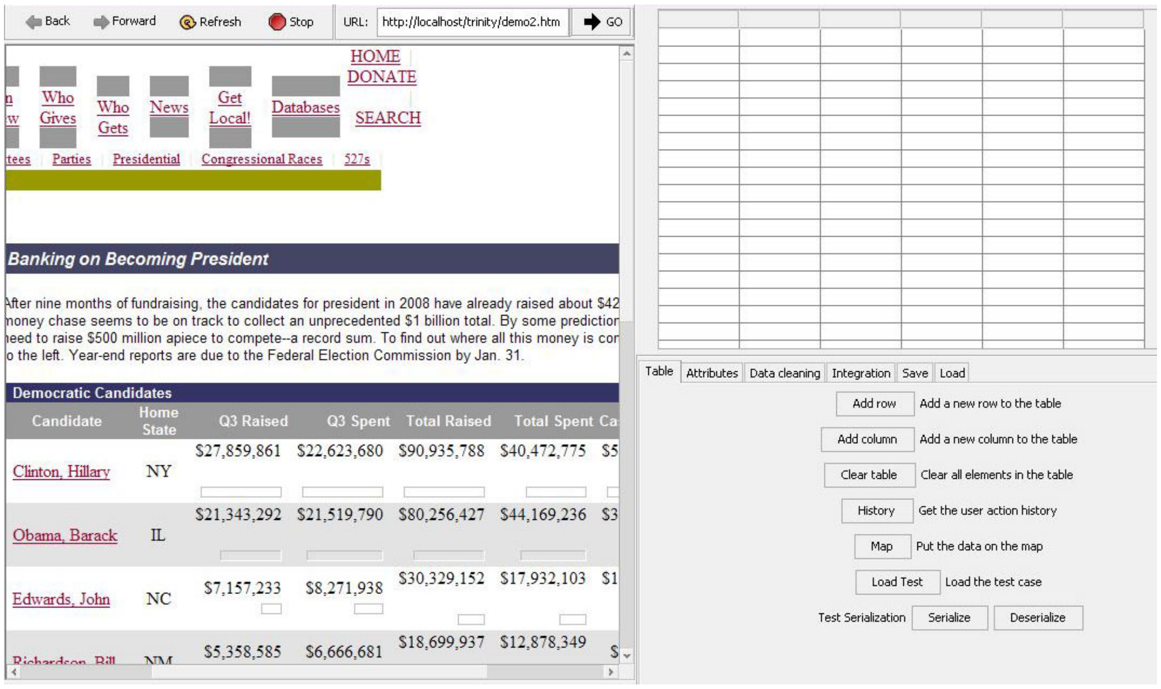
In this thesis, the coverage will be for the *1:1* case with non-dynamic data, since the contribution of the thesis is not a new source modeling technique, but the framework that ties Mashup building steps together. Research work focusing on solving the source modeling problem with sophisticated techniques can be found at [51, 18].

4.2 Approach

In this section, we first define an example task. Then, a snapshot of interactions between a user and Karma is shown. Finally, we discuss technical details that enable those interactions.

4.2.1 Case Study Walkthrough

In this chapter, we will use a new example from a web source that contains the information about a political campaign, but the structure of the problem is similar to the music example from the previous chapter. Figure 4.2 shows the web source about the political donation for the 2008 presidential primary campaign in the third quarter. The task for this case study is to retrieve the data and assign an attribute name for each column in the table. Since we already discussed data retrieval, we will assume that the user has already retrieved the data and the state of the table looks like Figure 4.3.



Banking on Becoming President

After nine months of fundraising, the candidates for president in 2008 have already raised about $42
noney chase seems to be on track to collect an unprecedented $1 billion total. By some prediction
need to raise $500 million apiece to compete--a record sum. To find out where all this money is cor
o the left. Year-end reports are due to the Federal Election Commission by Jan. 31.

Democratic Candidates

Candidate	Home State	Q3 Raised	Q3 Spent	Total Raised	Total Spent	Ca
Clinton, Hillary	NY	$27,859,861	$22,623,680	$90,935,788	$40,472,775	$5
Obama, Barack	IL	$21,343,292	$21,519,790	$80,256,427	$44,169,236	$3
Edwards, John	NC	$7,157,233	$8,271,938	$30,329,152	$17,932,103	$1
Richardson, Bill	NM	$5,358,585	$6,666,681	$18,699,937	$12,878,349	$

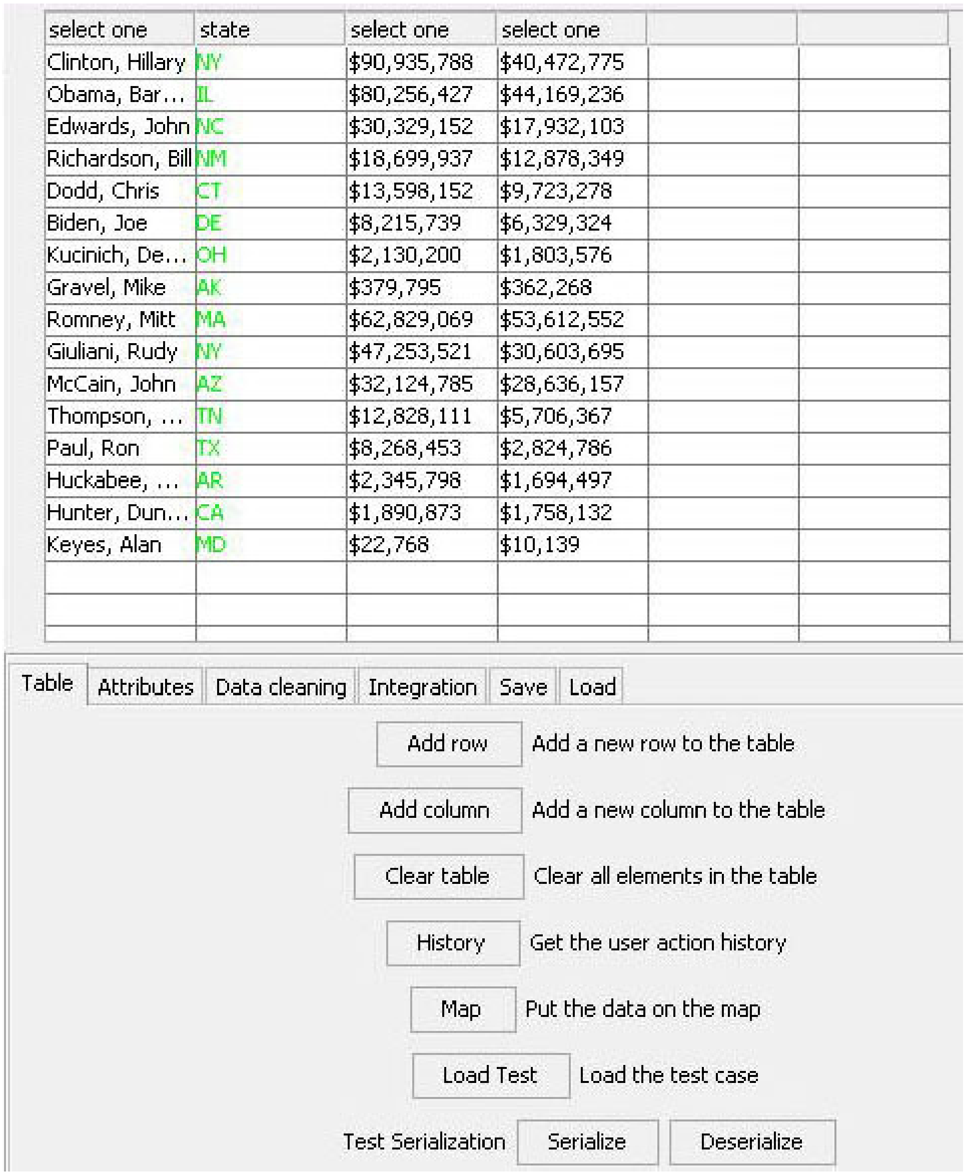
Add row	Add a new row to the table
Add column	Add a new column to the table
Clear table	Clear all elements in the table
History	Get the user action history
Map	Put the data on the map
Load Test	Load the test case
Test Serialization	Serialize Deserialize

Figure 4.2: Task: retrieving the political donation data for the 2008 presidential primary campaign and assigning an attribute to each data column.

While the source modeling problem tries to map a new source to existing sources, from the Karma user's point of view, she does not have to know anything about existing sources. All she needs to do is recognize that there are four possible interaction scenarios (our case study covers the first three cases):

1. Karma determines the attribute mapping and sets the attribute name for a column automatically.

2. Karma cannot determine an attribute name and the user would have to select a possible attribute match from a list of existing attributes from the database.



select one	state	select one	select one		
Clinton, Hillary	NY	$90,935,788	$40,472,775		
Obama, Bar...	IL	$80,256,427	$44,169,236		
Edwards, John	NC	$30,329,152	$17,932,103		
Richardson, Bill	NM	$18,699,937	$12,878,349		
Dodd, Chris	CT	$13,598,152	$9,723,278		
Biden, Joe	DE	$8,215,739	$6,329,324		
Kucinich, De...	OH	$2,130,200	$1,803,576		
Gravel, Mike	AK	$379,795	$362,268		
Romney, Mitt	MA	$62,829,069	$53,612,552		
Giuliani, Rudy	NY	$47,253,521	$30,603,695		
McCain, John	AZ	$32,124,785	$28,636,157		
Thompson, ...	TN	$12,828,111	$5,706,367		
Paul, Ron	TX	$8,268,453	$2,824,786		
Huckabee, ...	AR	$2,345,798	$1,694,497		
Hunter, Dun...	CA	$1,890,873	$1,758,132		
Keyes, Alan	MD	$22,768	$10,139		

Figure 4.3: The state of the table after finishing the data retrieval phrase.

3. Karma cannot determine the attribute name for a column and there is no matching attribute existed in the database. As a result, the user has to define an attribute manually.

4. Karma found multiple possible matches. For example, the data *Stephen* could be map to the attribute *firstname* or the attribute *lastname*. In this case, the user would need to select one from the list based on her own judgement.

As soon as the data is populated in a column, Karma check the databases to see if it can assign any attribute for that column. For example, Karma sets the second column attribute to *state* in Figure 4.3. For the column that Karma is not confident or cannot find any match, the attribute name will be set to *select one* to prompt the user to select an attribute.

The user can start assigning an attribute for each column by clicking the *Attributes* tab under the table as shown in Figure 4.4. The *Attributes* tab section is divided into three areas. The left area provides a drop down menu that lets the user select which column to work on. The leftmost column is designated as *Column 0*. The middle area lets the user manually type in a new attribute name or picks from available attributes. The right area contains an *update* button that the user can click to update the attribute name of the chosen column in the table.

Figure 4.5 to Figure 4.7 show how the user assigns the attribute for the first column (case1). When the user clicks on a text field input above the *suggestion* box, Karma shows every attribute that it knows of from the database. The user can filter this suggestion list by entering a partial text (i.e., *c*) into the text field input as shown in Figure 4.6.

By narrowing down available choices, Karma hopes that the user will be able to pick the correct attribute from the list. In this example, let us assume that the user picks *candidate* as the attribute for the first column by selecting it from the suggestion list and pressing the update button in Figure 4.7.

Figure 4.8 shows the attribute suggestion list for the second column. Note that the attribute has already been set. The lighter font represents the fact that there is a data overlapping (case 2) between new data elements and existing data elements in the database; In this example, Karma found that 16 new data elements exist in some tables in the database under the attribute name *state*. While the attribute name has already been set, it is also possible for the user to override Karma's decision by selecting a new attribute name.

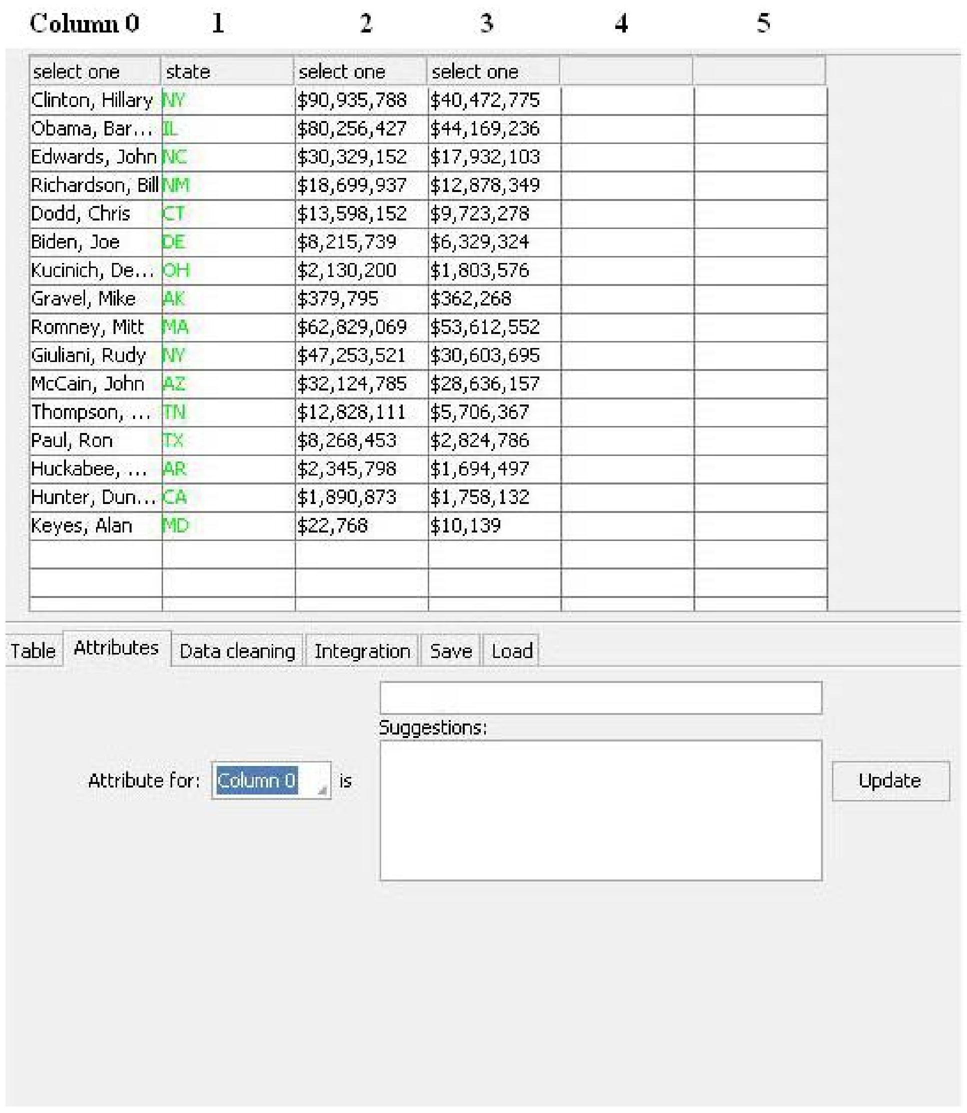
Figure 4.9 and 4.10 show the third case where Karma cannot find any match, and existing attributes cannot be mapped to new data elements. In this case, the user has to manually define the attribute herself.

4.2.2 Computing a Candidate Set

Whenever new data elements are populated in the table, Karma analyzes those new entries by comparing them with data elements that exist in databases to generate a possible candidate set for each column's attribute. To compute a candidate set used for suggestion, the following constraint is used. Let:

1. V: a set of values from the new column

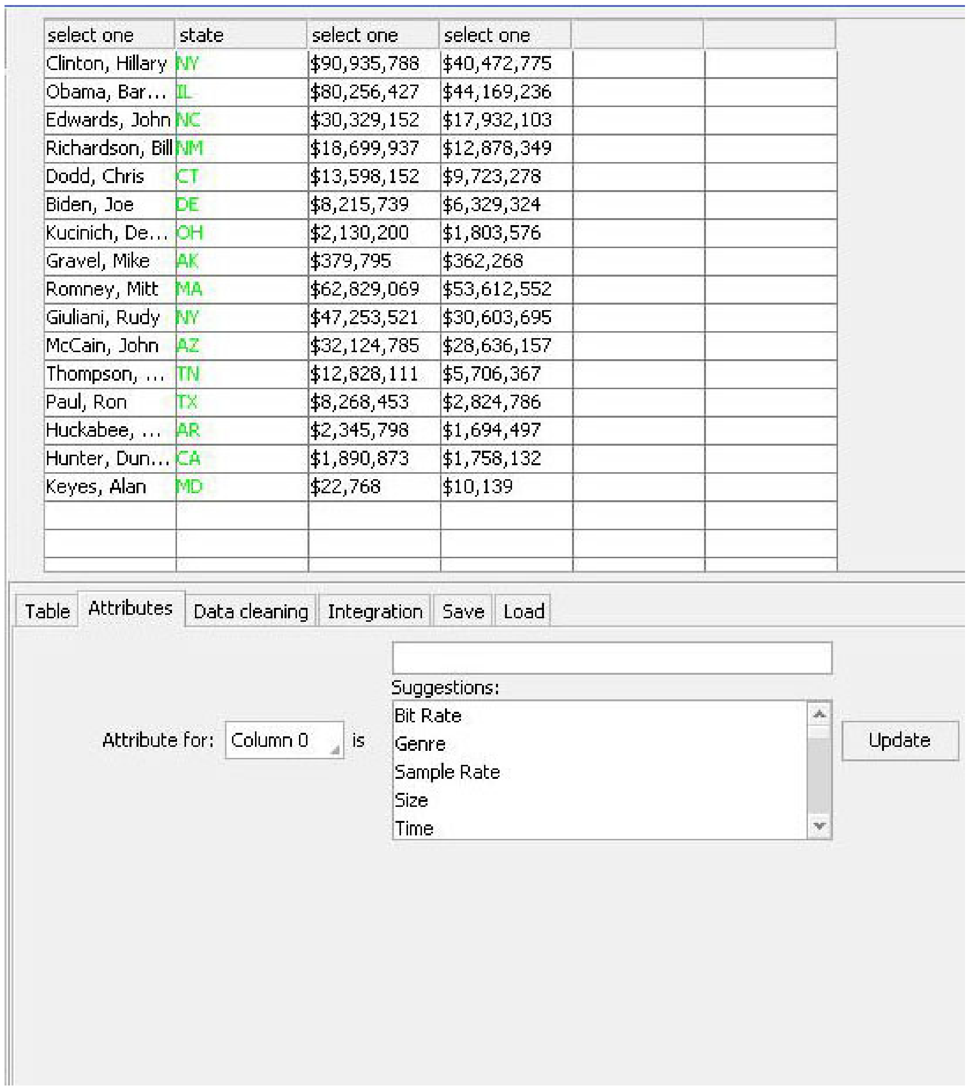
2. S: a set of all available data sources in the database



Column 0	1	2	3	4	5
select one	state	select one	select one		
Clinton, Hillary	NY	$90,935,788	$40,472,775		
Obama, Bar...	IL	$80,256,427	$44,169,236		
Edwards, John	NC	$30,329,152	$17,932,103		
Richardson, Bill	NM	$18,699,937	$12,878,349		
Dodd, Chris	CT	$13,598,152	$9,723,278		
Biden, Joe	DE	$8,215,739	$6,329,324		
Kucinich, De...	OH	$2,130,200	$1,803,576		
Gravel, Mike	AK	$379,795	$362,268		
Romney, Mitt	MA	$62,829,069	$53,612,552		
Giuliani, Rudy	NY	$47,253,521	$30,603,695		
McCain, John	AZ	$32,124,785	$28,636,157		
Thompson, ...	TN	$12,828,111	$5,706,367		
Paul, Ron	TX	$8,268,453	$2,824,786		
Huckabee, ...	AR	$2,345,798	$1,694,497		
Hunter, Dun...	CA	$1,890,873	$1,758,132		
Keyes, Alan	MD	$22,768	$10,139		

Attribute for: Column 0 is

Figure 4.4: The user can assign attribute by selecting the *Attributes* tab.

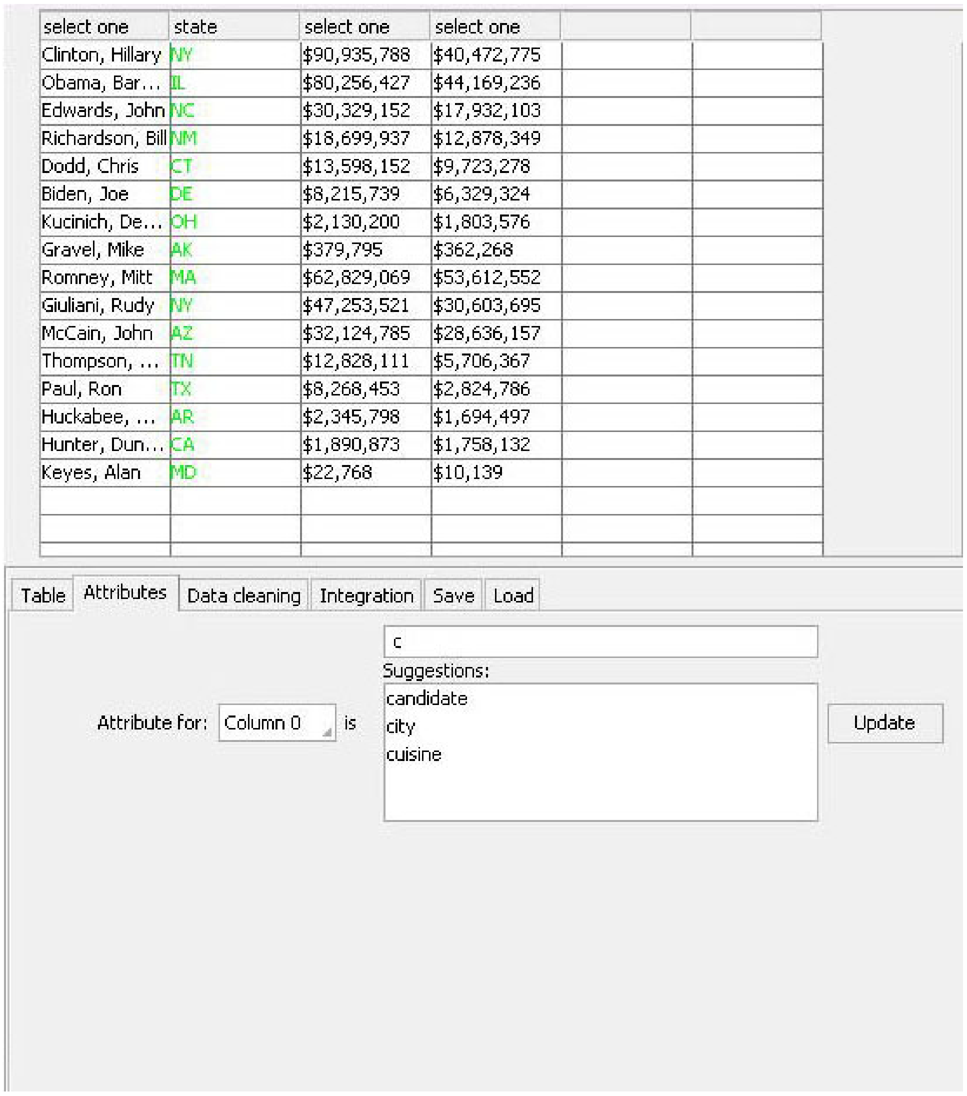


select one	state	select one	select one		
Clinton, Hillary	NY	$90,935,788	$40,472,775		
Obama, Bar...	IL	$80,256,427	$44,169,236		
Edwards, John	NC	$30,329,152	$17,932,103		
Richardson, Bill	NM	$18,699,937	$12,878,349		
Dodd, Chris	CT	$13,598,152	$9,723,278		
Biden, Joe	DE	$8,215,739	$6,329,324		
Kucinich, De...	OH	$2,130,200	$1,803,576		
Gravel, Mike	AK	$379,795	$362,268		
Romney, Mitt	MA	$62,829,069	$53,612,552		
Giuliani, Rudy	NY	$47,253,521	$30,603,695		
McCain, John	AZ	$32,124,785	$28,636,157		
Thompson, ...	TN	$12,828,111	$5,706,367		
Paul, Ron	TX	$8,268,453	$2,824,786		
Huckabee, ...	AR	$2,345,798	$1,694,497		
Hunter, Dun...	CA	$1,890,873	$1,758,132		
Keyes, Alan	MD	$22,768	$10,139		

Attribute for: Column 0 is

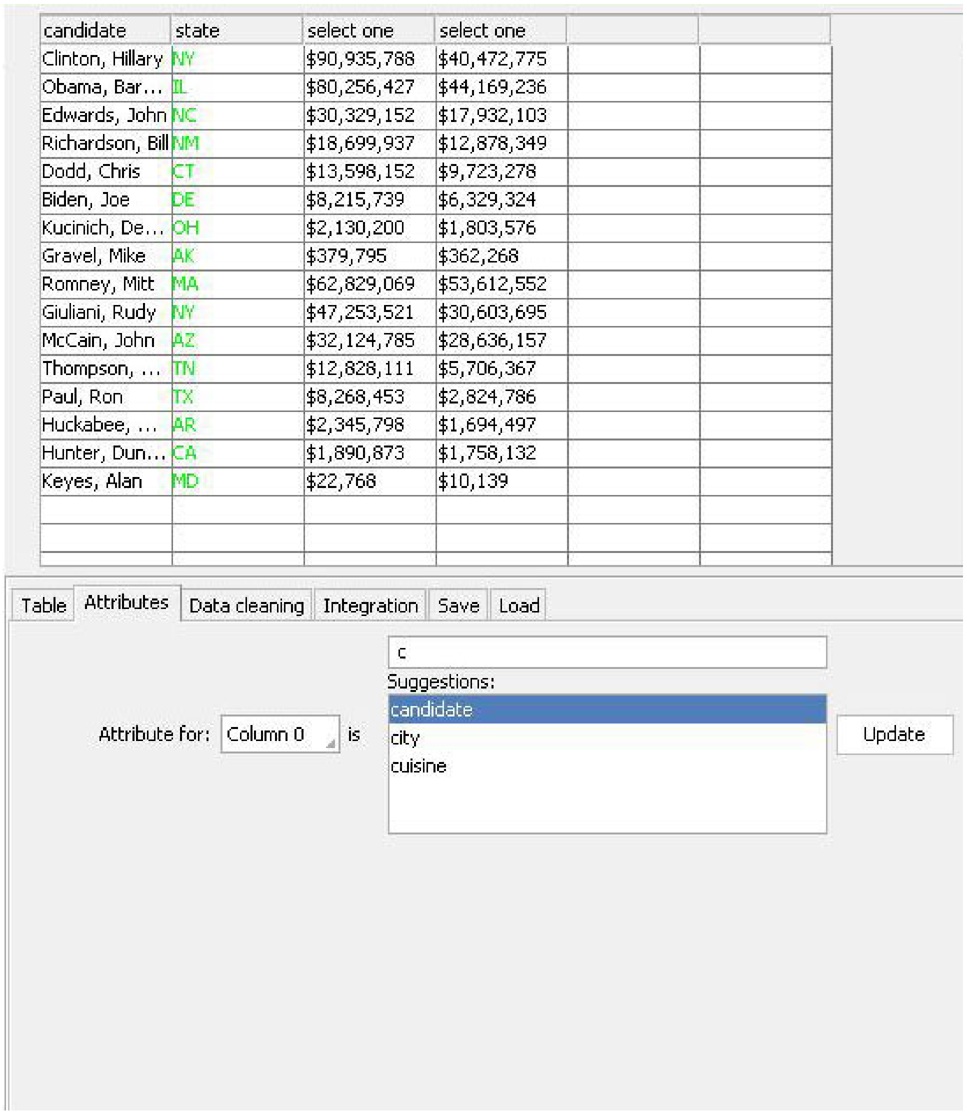
Suggestions:
Bit Rate
Genre
Sample Rate
Size
Time

Figure 4.5: The user can pick an attribute from existing attributes in the database when there is no data overlap.



select one	state	select one	select one		
Clinton, Hillary	NY	$90,935,788	$40,472,775		
Obama, Bar...	IL	$80,256,427	$44,169,236		
Edwards, John	NC	$30,329,152	$17,932,103		
Richardson, Bill	NM	$18,699,937	$12,878,349		
Dodd, Chris	CT	$13,598,152	$9,723,278		
Biden, Joe	DE	$8,215,739	$6,329,324		
Kucinich, De...	OH	$2,130,200	$1,803,576		
Gravel, Mike	AK	$379,795	$362,268		
Romney, Mitt	MA	$62,829,069	$53,612,552		
Giuliani, Rudy	NY	$47,253,521	$30,603,695		
McCain, John	AZ	$32,124,785	$28,636,157		
Thompson, ...	TN	$12,828,111	$5,706,367		
Paul, Ron	TX	$8,268,453	$2,824,786		
Huckabee, ...	AR	$2,345,798	$1,694,497		
Hunter, Dun...	CA	$1,890,873	$1,758,132		
Keyes, Alan	MD	$22,768	$10,139		

Figure 4.6: The user can filter the list of existing attributes by typing in a partial attribute name.

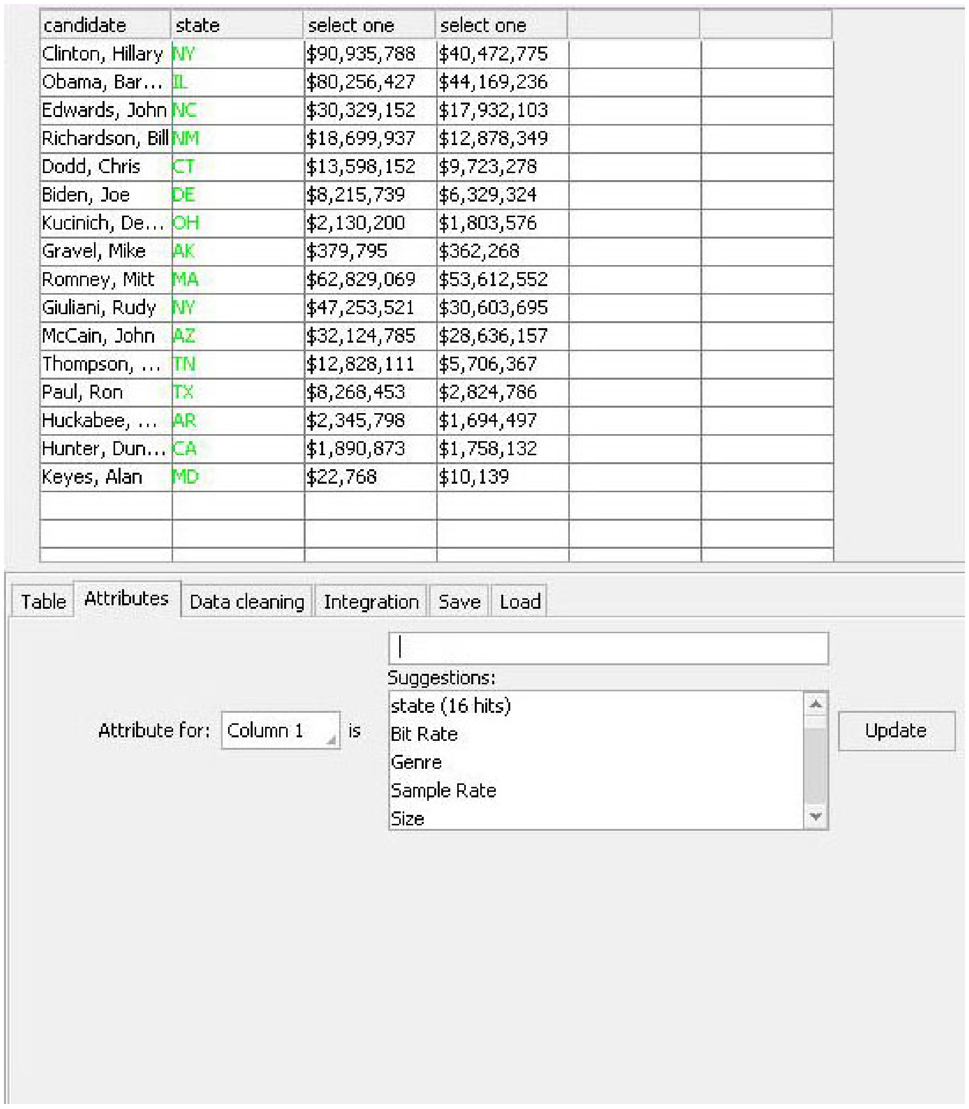


candidate	state	select one	select one		
Clinton, Hillary	NY	$90,935,788	$40,472,775		
Obama, Bar...	IL	$80,256,427	$44,169,236		
Edwards, John	NC	$30,329,152	$17,932,103		
Richardson, Bill	NM	$18,699,937	$12,878,349		
Dodd, Chris	CT	$13,598,152	$9,723,278		
Biden, Joe	DE	$8,215,739	$6,329,324		
Kucinich, De...	OH	$2,130,200	$1,803,576		
Gravel, Mike	AK	$379,795	$362,268		
Romney, Mitt	MA	$62,829,069	$53,612,552		
Giuliani, Rudy	NY	$47,253,521	$30,603,695		
McCain, John	AZ	$32,124,785	$28,636,157		
Thompson, ...	TN	$12,828,111	$5,706,367		
Paul, Ron	TX	$8,268,453	$2,824,786		
Huckabee, ...	AR	$2,345,798	$1,694,497		
Hunter, Dun...	CA	$1,890,873	$1,758,132		
Keyes, Alan	MD	$22,768	$10,139		

Attribute for: Column 0 is c

Suggestions:
candidate
city
cuisine

Figure 4.7: The user can set an attribute by selecting an attribute from the suggestion list and clicking *Update*.



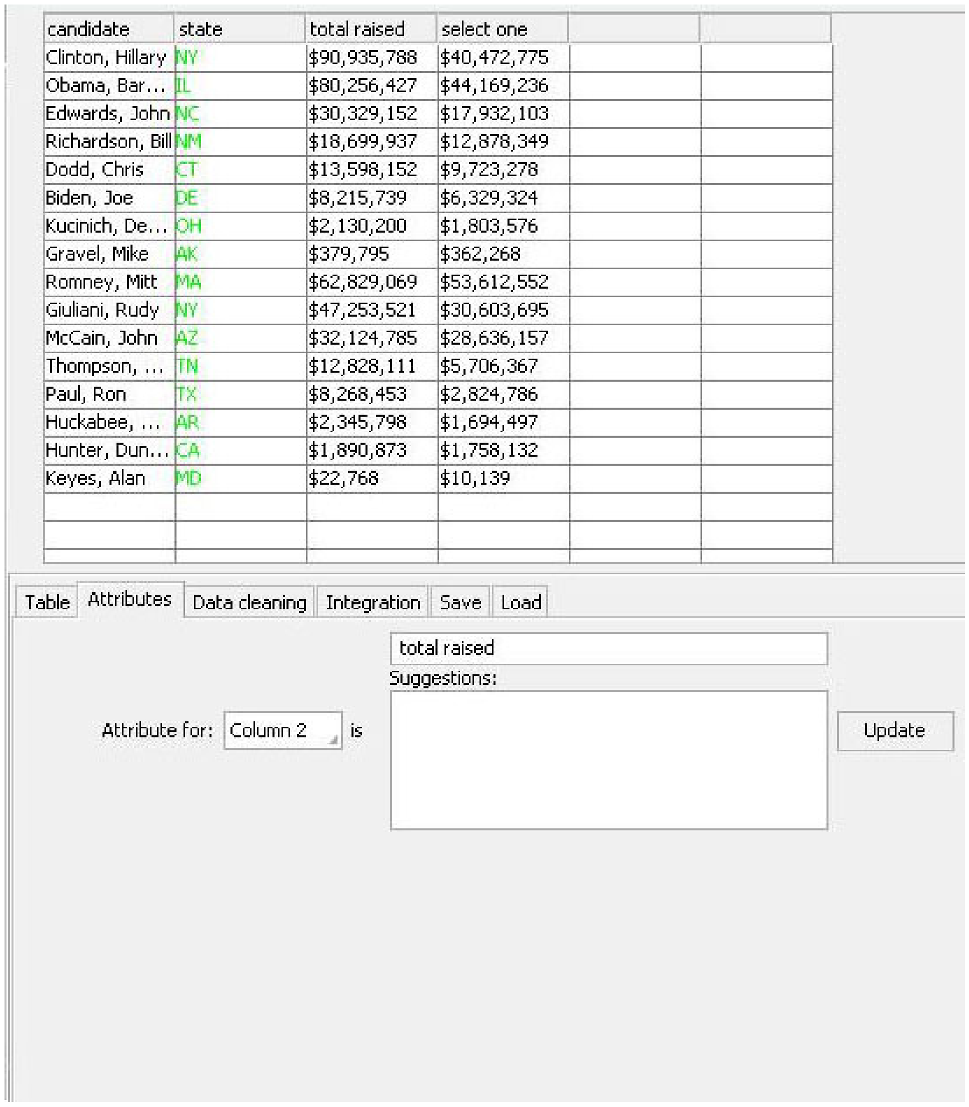
candidate	state	select one	select one		
Clinton, Hillary	NY	$90,935,788	$40,472,775		
Obama, Bar...	IL	$80,256,427	$44,169,236		
Edwards, John	NC	$30,329,152	$17,932,103		
Richardson, Bill	NM	$18,699,937	$12,878,349		
Dodd, Chris	CT	$13,598,152	$9,723,278		
Biden, Joe	DE	$8,215,739	$6,329,324		
Kucinich, De...	OH	$2,130,200	$1,803,576		
Gravel, Mike	AK	$379,795	$362,268		
Romney, Mitt	MA	$62,829,069	$53,612,552		
Giuliani, Rudy	NY	$47,253,521	$30,603,695		
McCain, John	AZ	$32,124,785	$28,636,157		
Thompson, ...	TN	$12,828,111	$5,706,367		
Paul, Ron	TX	$8,268,453	$2,824,786		
Huckabee, ...	AR	$2,345,798	$1,694,497		
Hunter, Dun...	CA	$1,890,873	$1,758,132		
Keyes, Alan	MD	$22,768	$10,139		

Attribute for: Column 1 is

Suggestions:

- state (16 hits)
- Bit Rate
- Genre
- Sample Rate
- Size

Figure 4.8: The attribute suggestion for the second column. The list of suggested attributes will be shown at the top of the suggestion list along with the rest of the attributes from the database.

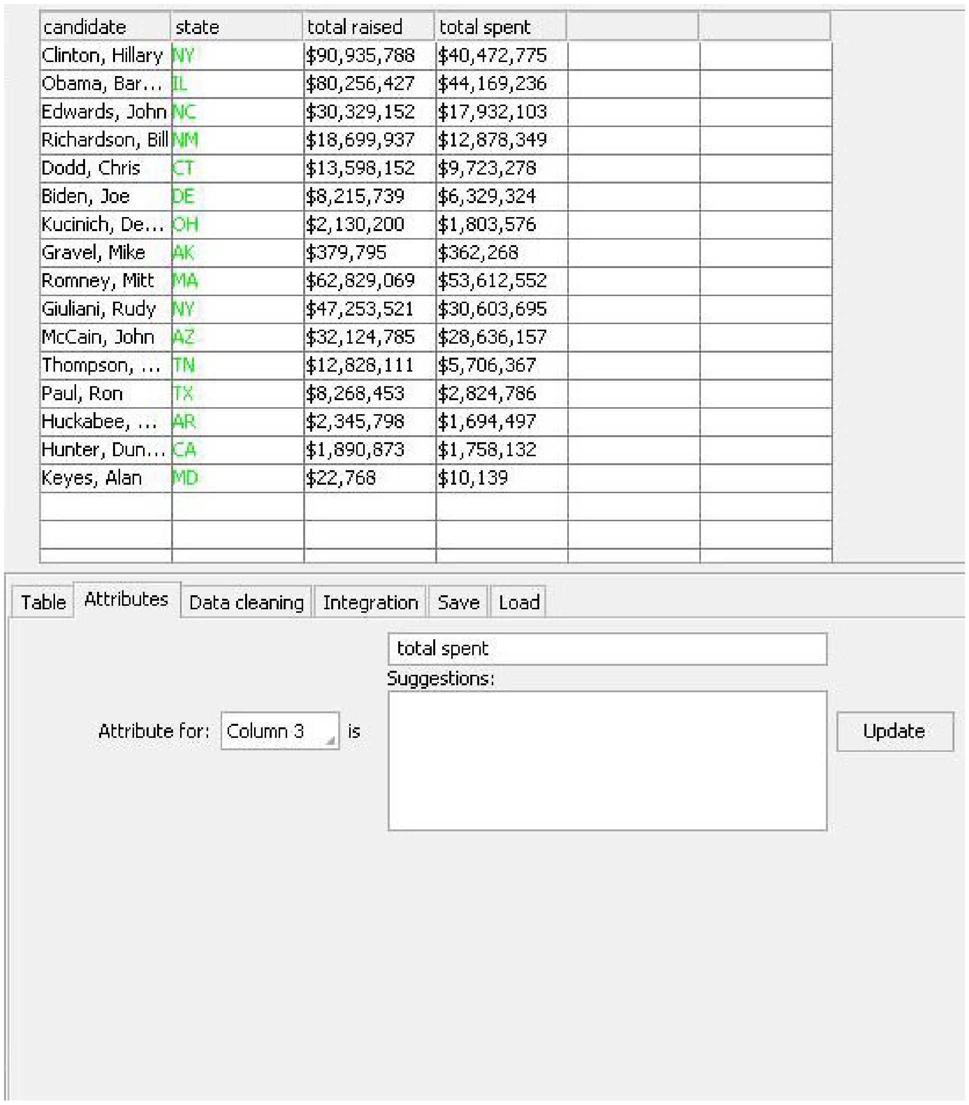


candidate	state	total raised	select one		
Clinton, Hillary	NY	$90,935,788	$40,472,775		
Obama, Bar...	IL	$80,256,427	$44,169,236		
Edwards, John	NC	$30,329,152	$17,932,103		
Richardson, Bill	NM	$18,699,937	$12,878,349		
Dodd, Chris	CT	$13,598,152	$9,723,278		
Biden, Joe	DE	$8,215,739	$6,329,324		
Kucinich, De...	OH	$2,130,200	$1,803,576		
Gravel, Mike	AK	$379,795	$362,268		
Romney, Mitt	MA	$62,829,069	$53,612,552		
Giuliani, Rudy	NY	$47,253,521	$30,603,695		
McCain, John	AZ	$32,124,785	$28,636,157		
Thompson, ...	TN	$12,828,111	$5,706,367		
Paul, Ron	TX	$8,268,453	$2,824,786		
Huckabee, ...	AR	$2,345,798	$1,694,497		
Hunter, Dun...	CA	$1,890,873	$1,758,132		
Keyes, Alan	MD	$22,768	$10,139		

Attribute for: Column 2 is total raised

Suggestions:

Figure 4.9: Manually assigning the new attribute *total raised*.



candidate	state	total raised	total spent		
Clinton, Hillary	NY	$90,935,788	$40,472,775		
Obama, Bar...	IL	$80,256,427	$44,169,236		
Edwards, John	NC	$30,329,152	$17,932,103		
Richardson, Bill	NM	$18,699,937	$12,878,349		
Dodd, Chris	CT	$13,598,152	$9,723,278		
Biden, Joe	DE	$8,215,739	$6,329,324		
Kucinich, De...	OH	$2,130,200	$1,803,576		
Gravel, Mike	AK	$379,795	$362,268		
Romney, Mitt	MA	$62,829,069	$53,612,552		
Giuliani, Rudy	NY	$47,253,521	$30,603,695		
McCain, John	AZ	$32,124,785	$28,636,157		
Thompson, ...	TN	$12,828,111	$5,706,367		
Paul, Ron	TX	$8,268,453	$2,824,786		
Huckabee, ...	AR	$2,345,798	$1,694,497		
Hunter, Dun...	CA	$1,890,873	$1,758,132		
Keyes, Alan	MD	$22,768	$10,139		

Attribute for: Column 3 is total spent

Suggestions:

Figure 4.10: Manually assigning the new attribute *total spent*.

3. *att(s)*: a procedure that returns the set of attributes from the source s where $s \in$ S

4. *val(a,s)*: a procedure that returns the set of values associated with the attribute a in the source s

The candidate set R is: $\{a|\forall a, s : a \in att(s) \wedge (val(a, s) \subset V)\}$. Figure 4.11 shows this constraint abstractly. For each value {x,y,z} in **V**, Karma tries to see if there is any overlapping data in the database or not. If there is any overlapping, Karma collects the overlapping attributes and put them in the candidate set. For example, x exists in **S1** and **S3** under the attribute *a1*, while y and z exist under the attribute *a6* from **S2**.

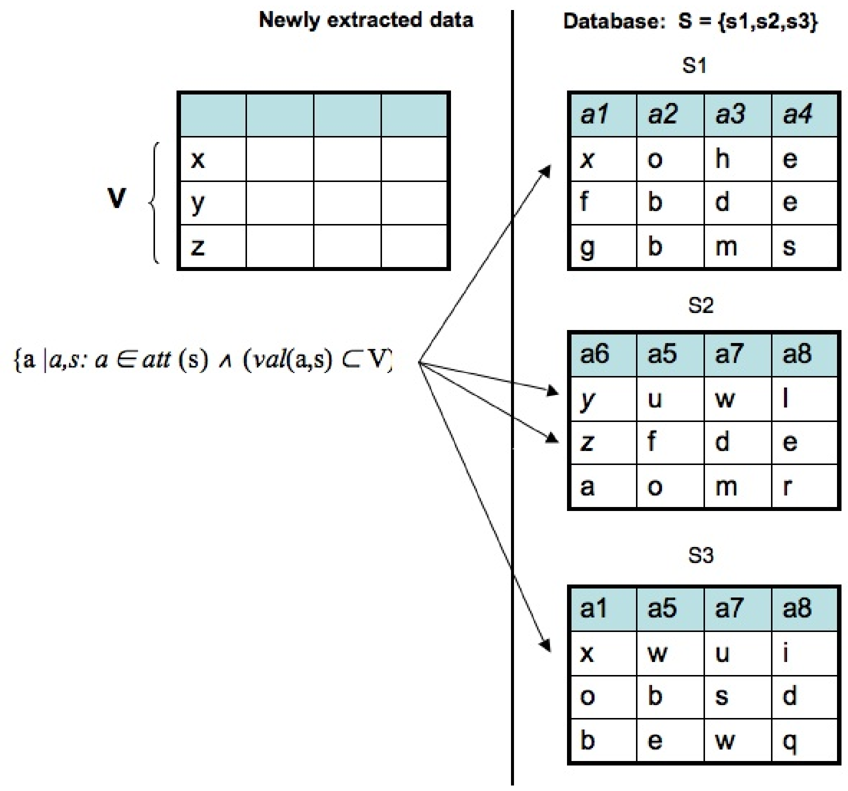


Figure 4.11: The abstract representation of how a candidate set is computed in Karma.

As a result, the suggestion list for this newly extracted data column will look like the following:

a1 (2 hits), a6 (2 hits), a2, a3, a4, a5, a7, a8

The overlapping attributes will be put at the top of the suggestion list, along with the number of occurrences of new data elements in the database. When there are multiple overlappings (case 4), Karma will not set the attribute for the new column automatically. If there is only one overlapping (case 1), then Karma will set the attribute for the new column automatically.

If Karma cannot find any matches, then the suggestion list will contain every attribute in the database arranged alphabetically:

a1, a2, a3, a4, a5, a6, a7, a8

Note that computing the candidate set requires us to lookup each new data token and find out if there is the same value in the database or not. To make the lookup more efficient in the database, we create a lookup table **L**. In this lookup table, we index all the values to construct a map $v \rightarrow \{(a,s)\}$ that maps the value (v) to the source (s) and the attribute (a) in that source where the value appear. Every time a new data table is saved into Karma's databases, its values are indexed into this lookup table. We also use this lookup table in the data integration phrase (section 6.3.1).

Since the user is retrieving data from the web site of her own choosing, we assume that the user will be able to select a correct attribute, if it exists in the database, or defines one. Note that existing Mashup building tools do not provide any guidance in terms of attribute selection; users of those systems have to define every attribute manually. One possible reason for not facilitating users in attribute selection might be because those systems never intend to use source modeling to help with the data cleaning problem or the data integration problem like Karma does.

4.3 Discussion

The source modeling approach shown in this chapter fits the Karma framework by leveraging historical data from the database. By locating overlapping data, Karma is able to suggest possible attribute mappings between the new source and existing sources. Also, as soon as the user puts the data into the table from the data retrieval phrase, the new data elements are pipelined into the source modeling problem phrase.

While the benefit of solving the source modeling problem might not be obvious from the user's point of view, this process will benefit the user later on during the data cleaning and data integration phrases. For example, when Karma can identify the column attribute with certainty, the data in that specific column is sent to the data cleaning phrase to detect misspellings (as shown in Figure 3.11), which will be discussed in the next chapter.

Chapter 5

Data Cleaning

This chapter is divided into three sections. The first section defines what data cleaning is and why it is important. The second section first shows how a user would interact with Karma to solve the data cleaning problem, and then describes the technical details behind those interactions. The third section elaborates on how my approach to the data cleaning problem fits the overall Karma framework.

5.1 Problem Definition

Data cleaning is a process of preparing data so that it will be ready for use in a task. While this definition seems to be very broad, it reflects the nature and difficulties associated with data cleaning. The set of data that can be used in one task might be unacceptable for use in another task. For example, the data of sensor performance that contains some errors may be acceptable for a backyard rocket project, but not a space shuttle project.

There are many ways to clean the data depending on the objectives. In this thesis, we segment data cleaning into four types:

- **Detecting discrepancy** involves locating the errant data. The error can be grammatical or contextual. An example of a grammatical error might be misspelling, such as *Wasington* instead of *Washington.* A contextual error requires understanding of the semantics to identify it. For example, a data element *200* under an attribute *age* is an error, because no human can live to two-hundred years.

- **Detecting duplication** involves finding and eliminating duplicated data points. This problem is often referred as a record linkage problem or a deduplication problem. For example, in Southern California restaurant data, *California Pizza Kitchen* and *C.P.K* are considered the same entity.

- **Extract, transform, load (ETL)** is a phrase used to describe the process of extracting data, transforming it, and loading it to another module. In many tasks, the data must be transformed to make it work with existing data or modules. Usually, the transformation function requires programming or scripting. The transformation can be static reformatting, such as naming format (e.g., *Clinton, Hillary* to *Hillary Clinton*). On the other hand, it can also includes dynamic transformation like currency conversion (e.g., US$1 to ThaiBaht 33).

- **Filtering** involves limiting the set of data. For example, a marketer might want to filter the census data to get the information about a specific demographic range (i.e., age 20-29) for a marketing campaign.

As shown above, data cleaning can be complicated and requires different approaches to address different problems that exist in the data. While Mashups covered in this thesis are aimed at casual use and users can tolerate some degrees of error, data cleaning is important for the fourth type of Mashup where data from two different sources are joined together. Figure 5.1 shows an example why data cleaning is important in building Mashups. To combine real estate listing with property tax, the address data from the real estate listing source must be fed into the HTML form input of the property tax source. If the address is wrong (e.g., misspelling), then the property tax source might not return any result, or incorrect results.

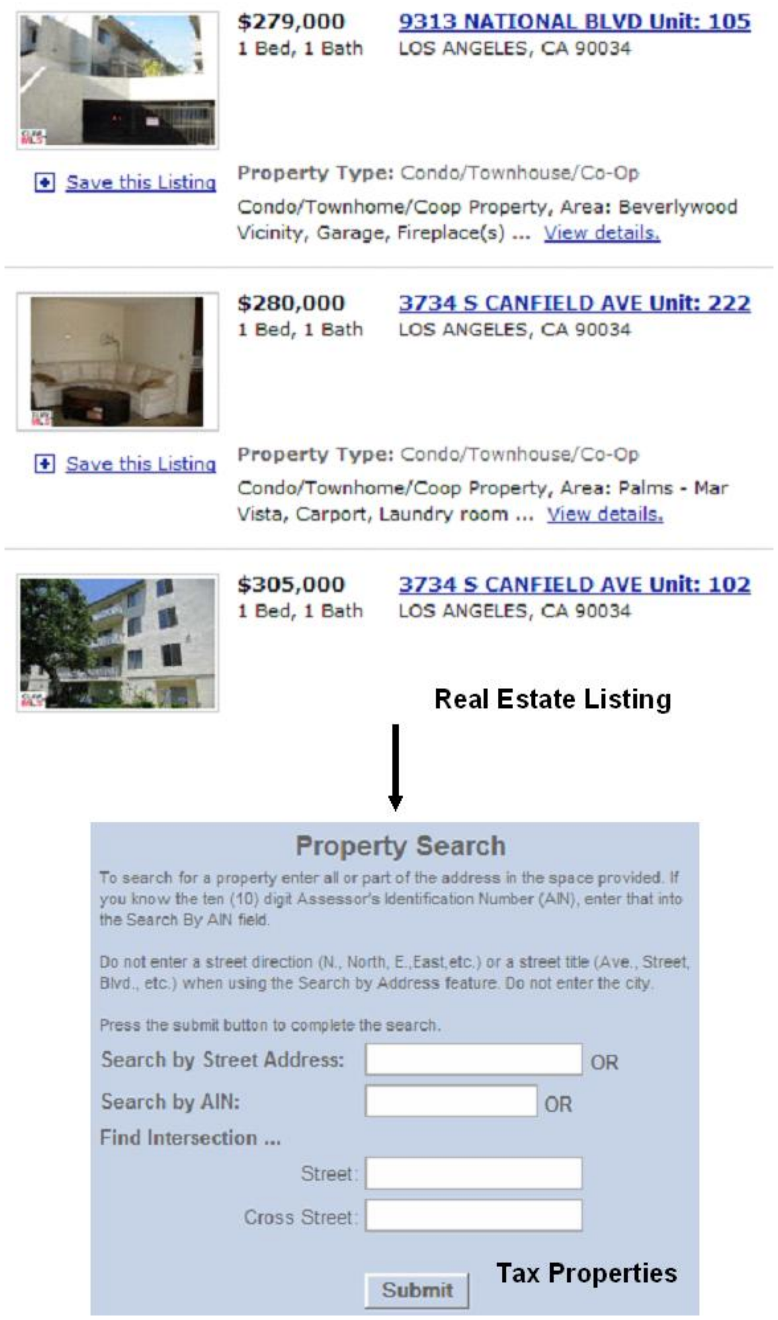
In this thesis, we selected the coverage for the data cleaning problem based on the trade off between functionality, complexity and usability. We demonstrate our approach by automatically detecting grammatical discrepancy and inducing transformation. While not currently addressed, detecting contextual discrepancy and detecting deduplication can also be implemented using the same approach without the loss of generality in the future.

5.2 Approach

In this section, we first define an example task. Then, we show a snapshot of interactions between a user and Karma. Finally, we discuss technical details that enable those interactions.

5.2.1 Case Study Walkthrough

Karma supports two kinds of data cleaning: detecting grammatical discrepancy and transformation. We will build on two examples discussed in previous chapters. Specifically, the first interaction, shown in Figure 5.2 through Figure 5.8, will show how the user can specify a transformation to clean the name format from the political contribution campaign example. The second interaction, shown in Figure 5.9 through Figure 5.13,


$279,000 9313 NATIONAL BLVD Unit: 105
1 Bed, 1 Bath LOS ANGELES, CA 90034

Save this Listing

Property Type: Condo/Townhouse/Co-Op

Condo/Townhome/Coop Property, Area: Beverlywood Vicinity, Garage, Fireplace(s) ... View details.

$280,000 3734 S CANFIELD AVE Unit: 222
1 Bed, 1 Bath LOS ANGELES, CA 90034

Save this Listing

Property Type: Condo/Townhouse/Co-Op

Condo/Townhome/Coop Property, Area: Palms - Mar Vista, Carport, Laundry room ... View details.

$305,000 3734 S CANFIELD AVE Unit: 102
1 Bed, 1 Bath LOS ANGELES, CA 90034

Property Search

To search for a property enter all or part of the address in the space provided. If you know the ten (10) digit Assessor's Identification Number (AIN), enter that into the Search By AIN field.

Do not enter a street direction (N., North, E.,East,etc.) or a street title (Ave., Street, Blvd., etc.) when using the Search by Address feature. Do not enter the city.

Press the submit button to complete the search.

Search by Street Address: OR
Search by AIN: OR
Find Intersection ...
Street:
Cross Street:
Submit

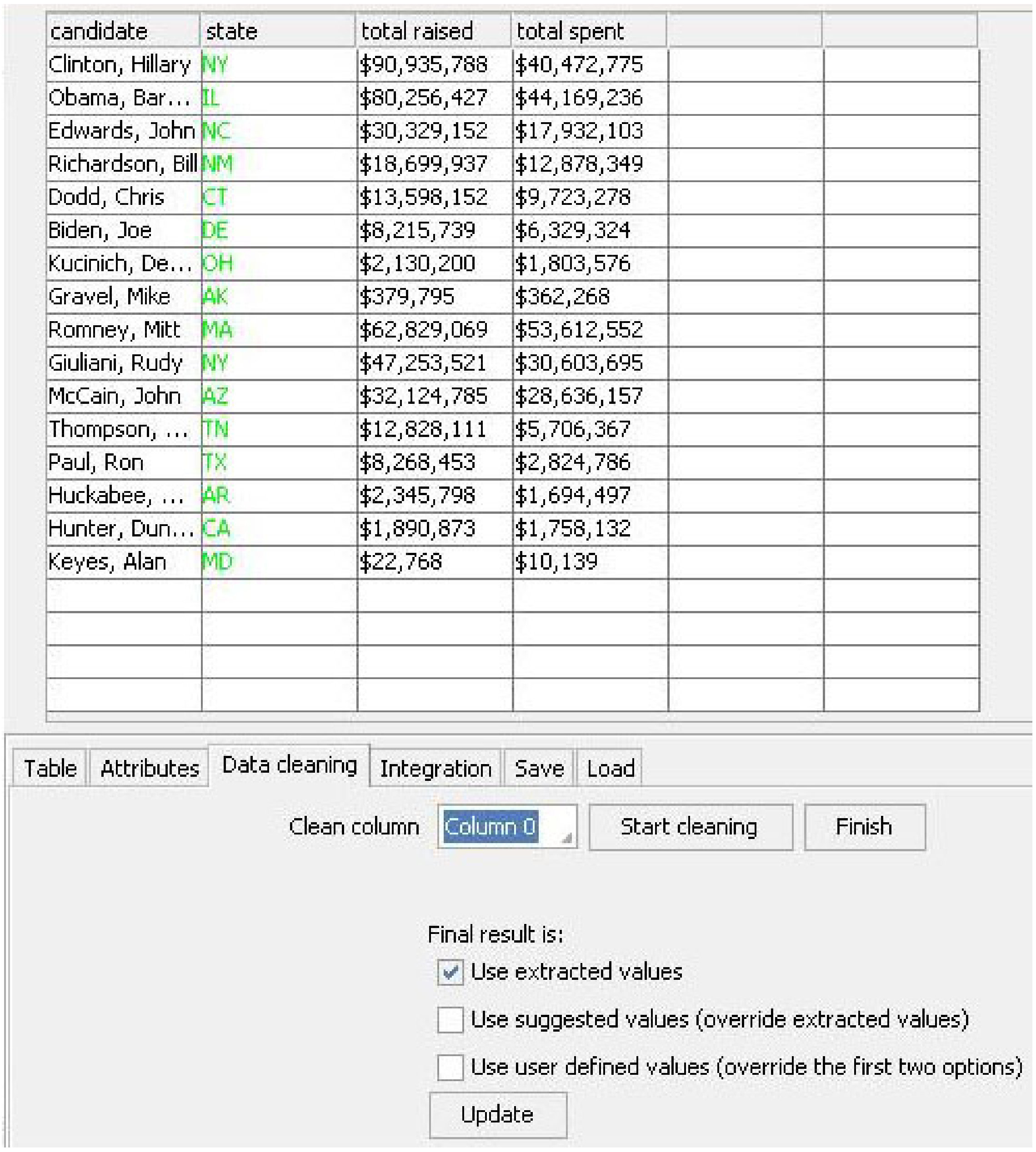
Figure 5.1: To retrieve the correct property tax, the address must be correct.

will show how Karma detects misspelling and makes suggestions to the user in the music example.

In our political campaign example from chapter 4, the naming format of the data is *Lastname, Firstname*. While this format is not wrong, the user might want to change it to *Firstname Lastname* to integrate it with other data. Assuming the user wants to clean the data, the user begins by selecting the *Data cleaning* tab under the table (Figure 5.2), choose the column that she wants to clean (i.e., *Column 0* in this case), and clicks *Start cleaning.*

Figure 5.3 shows Karma activating the data cleaning mode (by adding three new columns after the *candidate* column and graying out irrelevant columns. The three new columns are *suggest*, *user defined*, and *final*. The *suggest* column will contain Karma's suggestion for data replacement, if Karma thinks there is any misspelling. The role of the *suggest* column is not relevant in this example, but it will be discussed in the next example. The *user defined* column allows the user to clean the data by giving examples of what the data should look like, when cleaned. The *final* column will contain what the data should look like at the end.

In this example, the user will clean the data by giving an example. The user can click on any row under the *user defined* column and type in the cleaned value. Figure 5.4 shows the user typing in the text *Hillary Clinton* as the desired value. Once the user finishes typing in the value and presses enter, Karma will try to determine the transformation rule by looking at the original data and the specified data. If Karma can identify the transformation rule, Karma will apply that rule to the rest of the data as shown in Figure 5.5



candidate	state	total raised	total spent		
Clinton, Hillary	NY	$90,935,788	$40,472,775		
Obama, Bar...	IL	$80,256,427	$44,169,236		
Edwards, John	NC	$30,329,152	$17,932,103		
Richardson, Bill	NM	$18,699,937	$12,878,349		
Dodd, Chris	CT	$13,598,152	$9,723,278		
Biden, Joe	DE	$8,215,739	$6,329,324		
Kucinich, De...	OH	$2,130,200	$1,803,576		
Gravel, Mike	AK	$379,795	$362,268		
Romney, Mitt	MA	$62,829,069	$53,612,552		
Giuliani, Rudy	NY	$47,253,521	$30,603,695		
McCain, John	AZ	$32,124,785	$28,636,157		
Thompson, ...	TN	$12,828,111	$5,706,367		
Paul, Ron	TX	$8,268,453	$2,824,786		
Huckabee, ...	AR	$2,345,798	$1,694,497		
Hunter, Dun...	CA	$1,890,873	$1,758,132		
Keyes, Alan	MD	$22,768	$10,139		

Clean column: Column 0

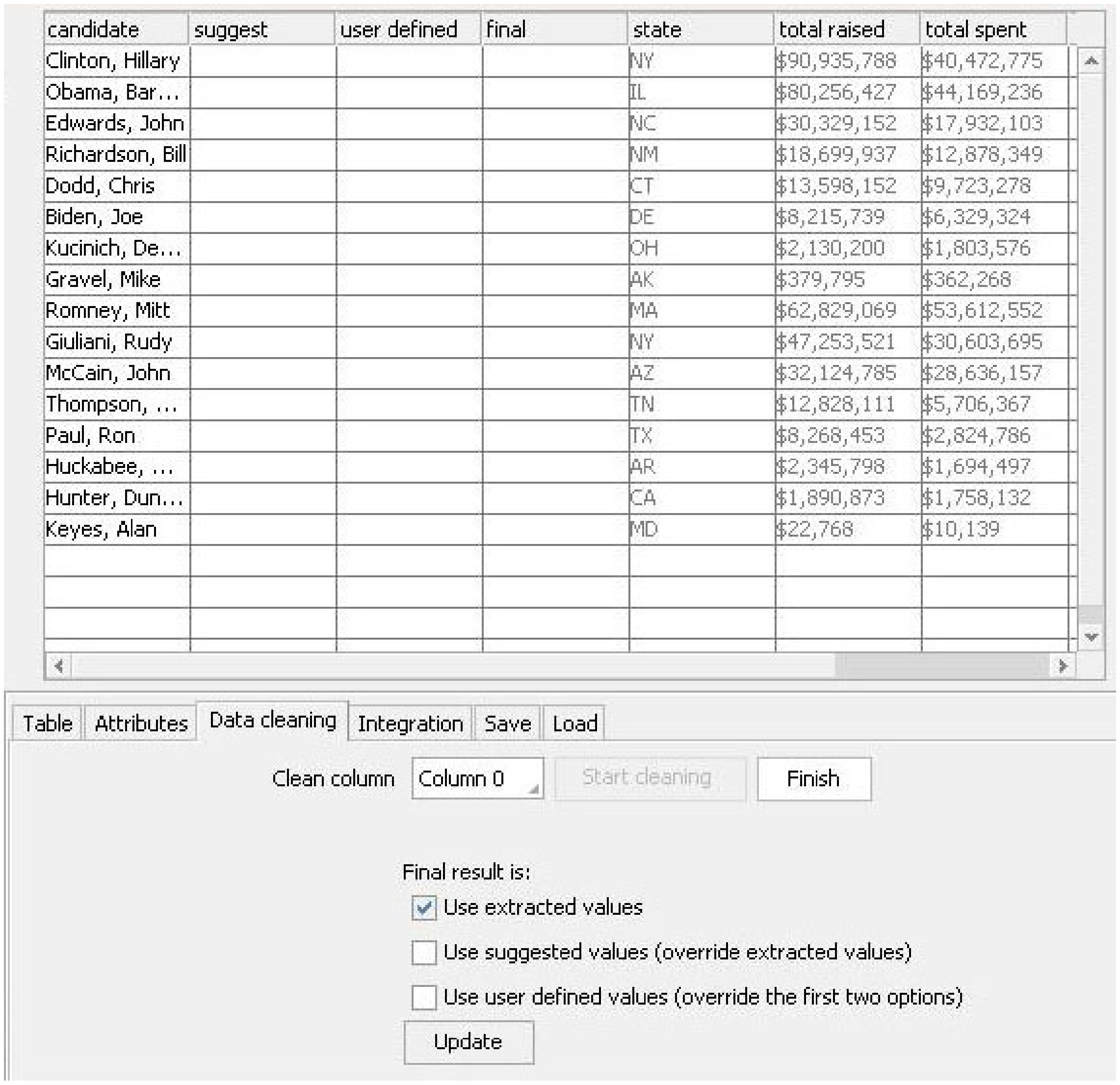
Final result is:

[✓] Use extracted values

[ ] Use suggested values (override extracted values)

[ ] Use user defined values (override the first two options)

Figure 5.2: The user can start the data cleaning process by selecting *Data cleaning* tab under the table.

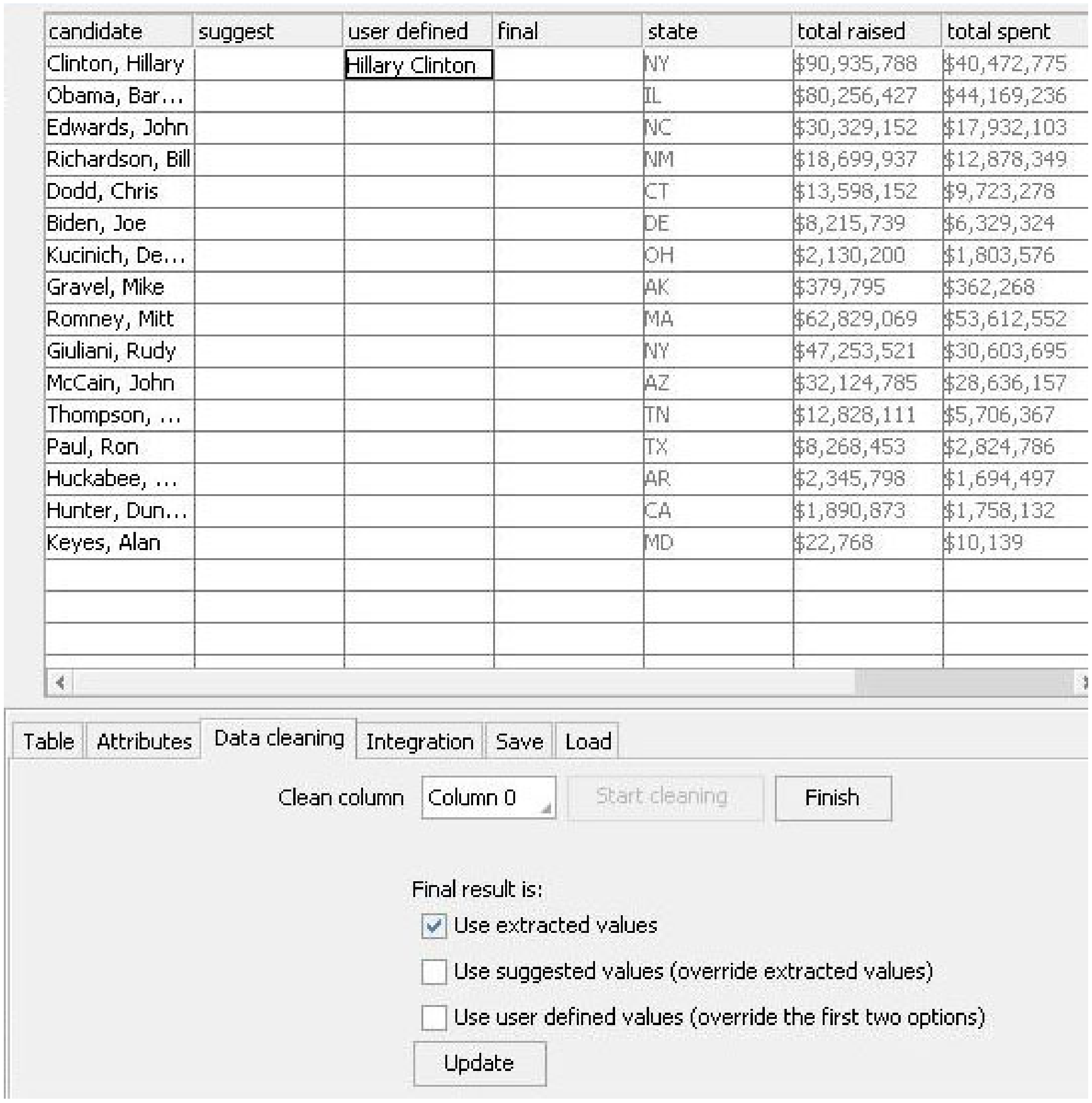


candidate	suggest	user defined	final	state	total raised	total spent
Clinton, Hillary				NY	$90,935,788	$40,472,775
Obama, Bar...				IL	$80,256,427	$44,169,236
Edwards, John				NC	$30,329,152	$17,932,103
Richardson, Bill				NM	$18,699,937	$12,878,349
Dodd, Chris				CT	$13,598,152	$9,723,278
Biden, Joe				DE	$8,215,739	$6,329,324
Kucinich, De...				OH	$2,130,200	$1,803,576
Gravel, Mike				AK	$379,795	$362,268
Romney, Mitt				MA	$62,829,069	$53,612,552
Giuliani, Rudy				NY	$47,253,521	$30,603,695
McCain, John				AZ	$32,124,785	$28,636,157
Thompson, ...				TN	$12,828,111	$5,706,367
Paul, Ron				TX	$8,268,453	$2,824,786
Huckabee, ...				AR	$2,345,798	$1,694,497
Hunter, Dun...				CA	$1,890,873	$1,758,132
Keyes, Alan				MD	$22,768	$10,139

Clean column: Column 0

Final result is:
[✓] Use extracted values
[ ] Use suggested values (override extracted values)
[ ] Use user defined values (override the first two options)

Figure 5.3: To clean the candidate column, the user chooses *Column 0* and clicks *Start cleaning*. Karma will create three new columns used for the data cleaning process and gray out other data columns.



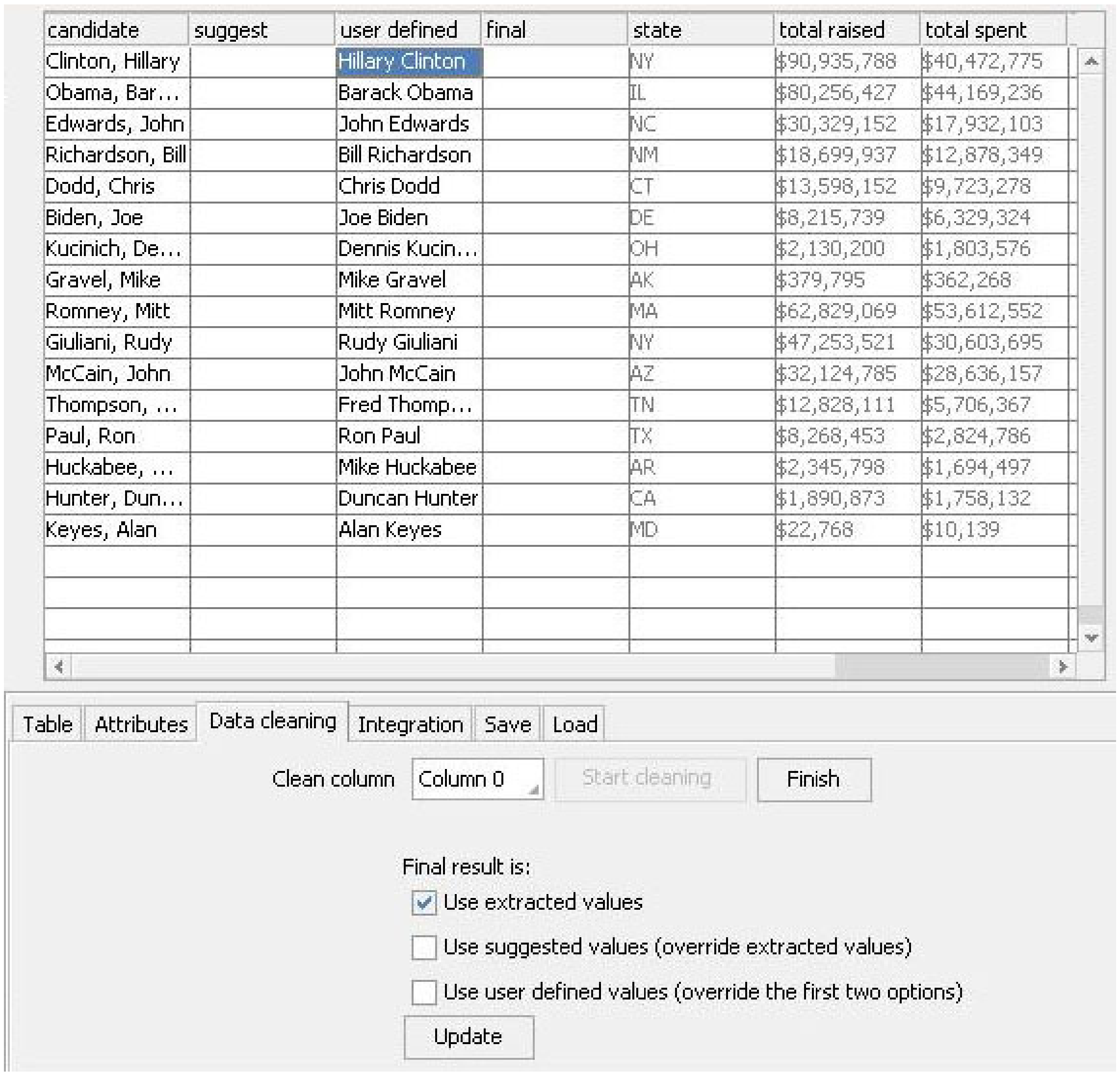
candidate	suggest	user defined	final	state	total raised	total spent
Clinton, Hillary		Hillary Clinton		NY	$90,935,788	$40,472,775
Obama, Bar…				IL	$80,256,427	$44,169,236
Edwards, John				NC	$30,329,152	$17,932,103
Richardson, Bill				NM	$18,699,937	$12,878,349
Dodd, Chris				CT	$13,598,152	$9,723,278
Biden, Joe				DE	$8,215,739	$6,329,324
Kucinich, De…				OH	$2,130,200	$1,803,576
Gravel, Mike				AK	$379,795	$362,268
Romney, Mitt				MA	$62,829,069	$53,612,552
Giuliani, Rudy				NY	$47,253,521	$30,603,695
McCain, John				AZ	$32,124,785	$28,636,157
Thompson, …				TN	$12,828,111	$5,706,367
Paul, Ron				TX	$8,268,453	$2,824,786
Huckabee, …				AR	$2,345,798	$1,694,497
Hunter, Dun…				CA	$1,890,873	$1,758,132
Keyes, Alan				MD	$22,768	$10,139

Clean column: Column 0

Final result is:

- [✓] Use extracted values
- [ ] Use suggested values (override extracted values)
- [ ] Use user defined values (override the first two options)

Figure 5.4: The user clean the data by typing in the end result that the user wants to see. In this case, the user gives an example of *Hillary Clinton* as the desired outcome.

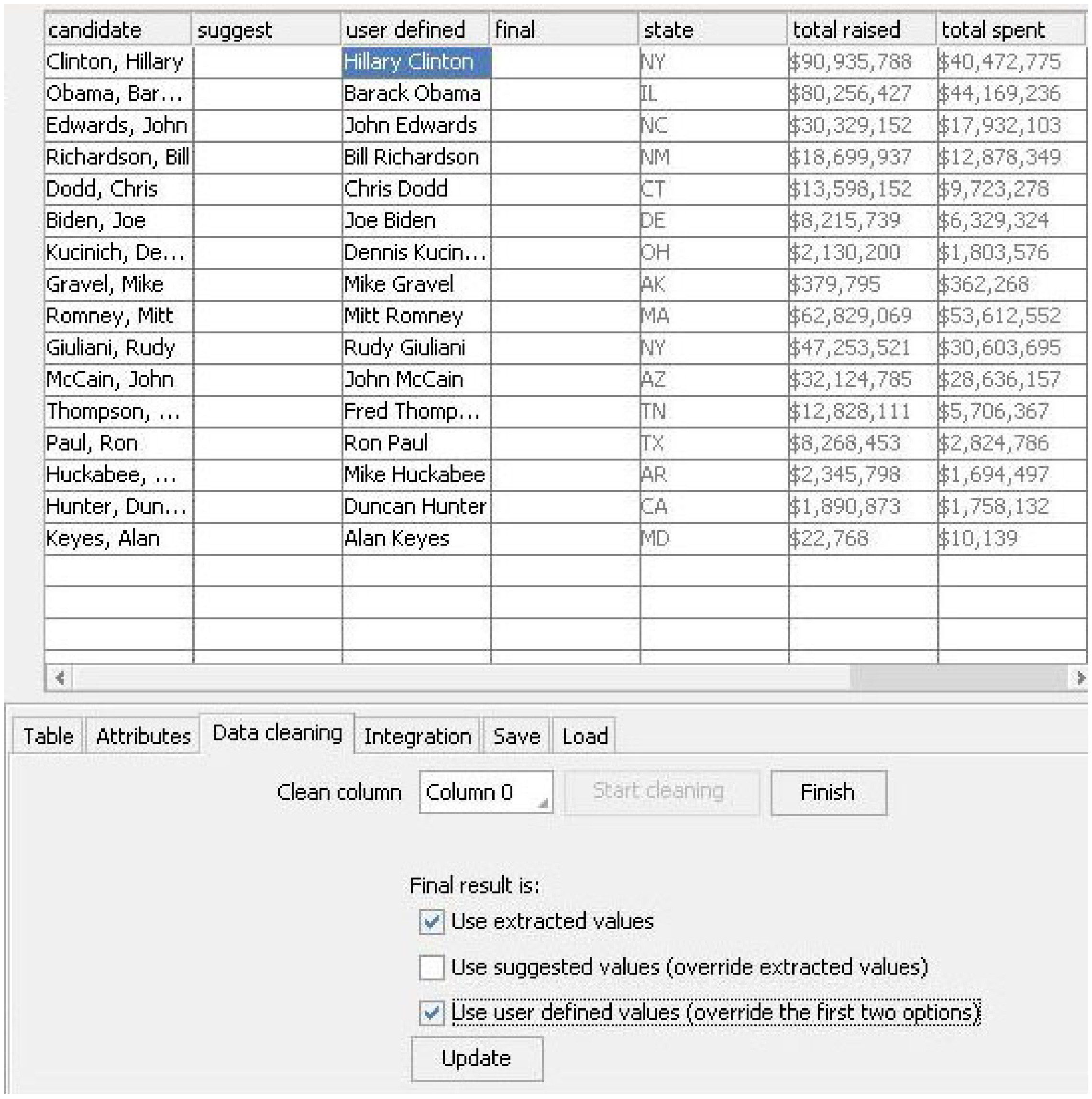


candidate	suggest	user defined	final	state	total raised	total spent
Clinton, Hillary		Hillary Clinton		NY	$90,935,788	$40,472,775
Obama, Bar...		Barack Obama		IL	$80,256,427	$44,169,236
Edwards, John		John Edwards		NC	$30,329,152	$17,932,103
Richardson, Bill		Bill Richardson		NM	$18,699,937	$12,878,349
Dodd, Chris		Chris Dodd		CT	$13,598,152	$9,723,278
Biden, Joe		Joe Biden		DE	$8,215,739	$6,329,324
Kucinich, De...		Dennis Kucin...		OH	$2,130,200	$1,803,576
Gravel, Mike		Mike Gravel		AK	$379,795	$362,268
Romney, Mitt		Mitt Romney		MA	$62,829,069	$53,612,552
Giuliani, Rudy		Rudy Giuliani		NY	$47,253,521	$30,603,695
McCain, John		John McCain		AZ	$32,124,785	$28,636,157
Thompson, ...		Fred Thomp...		TN	$12,828,111	$5,706,367
Paul, Ron		Ron Paul		TX	$8,268,453	$2,824,786
Huckabee, ...		Mike Huckabee		AR	$2,345,798	$1,694,497
Hunter, Dun...		Duncan Hunter		CA	$1,890,873	$1,758,132
Keyes, Alan		Alan Keyes		MD	$22,768	$10,139

Final result is:

- [✓] Use extracted values
- [ ] Use suggested values (override extracted values)
- [ ] Use user defined values (override the first two options)

Figure 5.5: Karma analyzes the given example and select the appropriate transformation rule to applied to the rest of the data in the user defined column.



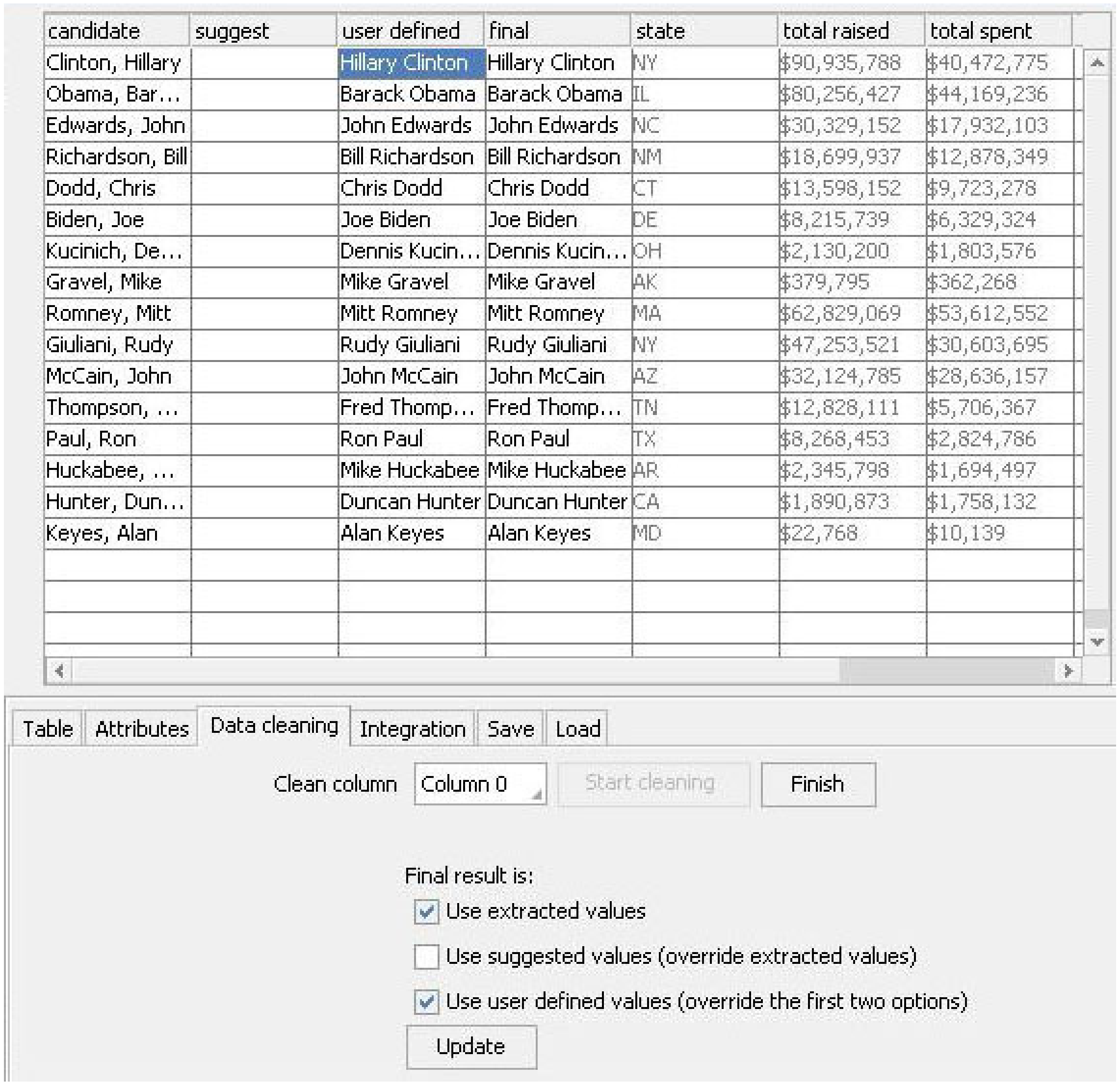
candidate	suggest	user defined	final	state	total raised	total spent
Clinton, Hillary		Hillary Clinton		NY	$90,935,788	$40,472,775
Obama, Bar...		Barack Obama		IL	$80,256,427	$44,169,236
Edwards, John		John Edwards		NC	$30,329,152	$17,932,103
Richardson, Bill		Bill Richardson		NM	$18,699,937	$12,878,349
Dodd, Chris		Chris Dodd		CT	$13,598,152	$9,723,278
Biden, Joe		Joe Biden		DE	$8,215,739	$6,329,324
Kucinich, De...		Dennis Kucin...		OH	$2,130,200	$1,803,576
Gravel, Mike		Mike Gravel		AK	$379,795	$362,268
Romney, Mitt		Mitt Romney		MA	$62,829,069	$53,612,552
Giuliani, Rudy		Rudy Giuliani		NY	$47,253,521	$30,603,695
McCain, John		John McCain		AZ	$32,124,785	$28,636,157
Thompson, ...		Fred Thomp...		TN	$12,828,111	$5,706,367
Paul, Ron		Ron Paul		TX	$8,268,453	$2,824,786
Huckabee, ...		Mike Huckabee		AR	$2,345,798	$1,694,497
Hunter, Dun...		Duncan Hunter		CA	$1,890,873	$1,758,132
Keyes, Alan		Alan Keyes		MD	$22,768	$10,139

Clean column: Column 0

Final result is:

- [✓] Use extracted values
- [ ] Use suggested values (override extracted values)
- [✓] Use user defined values (override the first two options)

Figure 5.6: The user tells Karma to use the *user defined* column as the final result by clicking the appropriate box. In this case, any cell under *user defined* that is not empty will override the original extracted values.



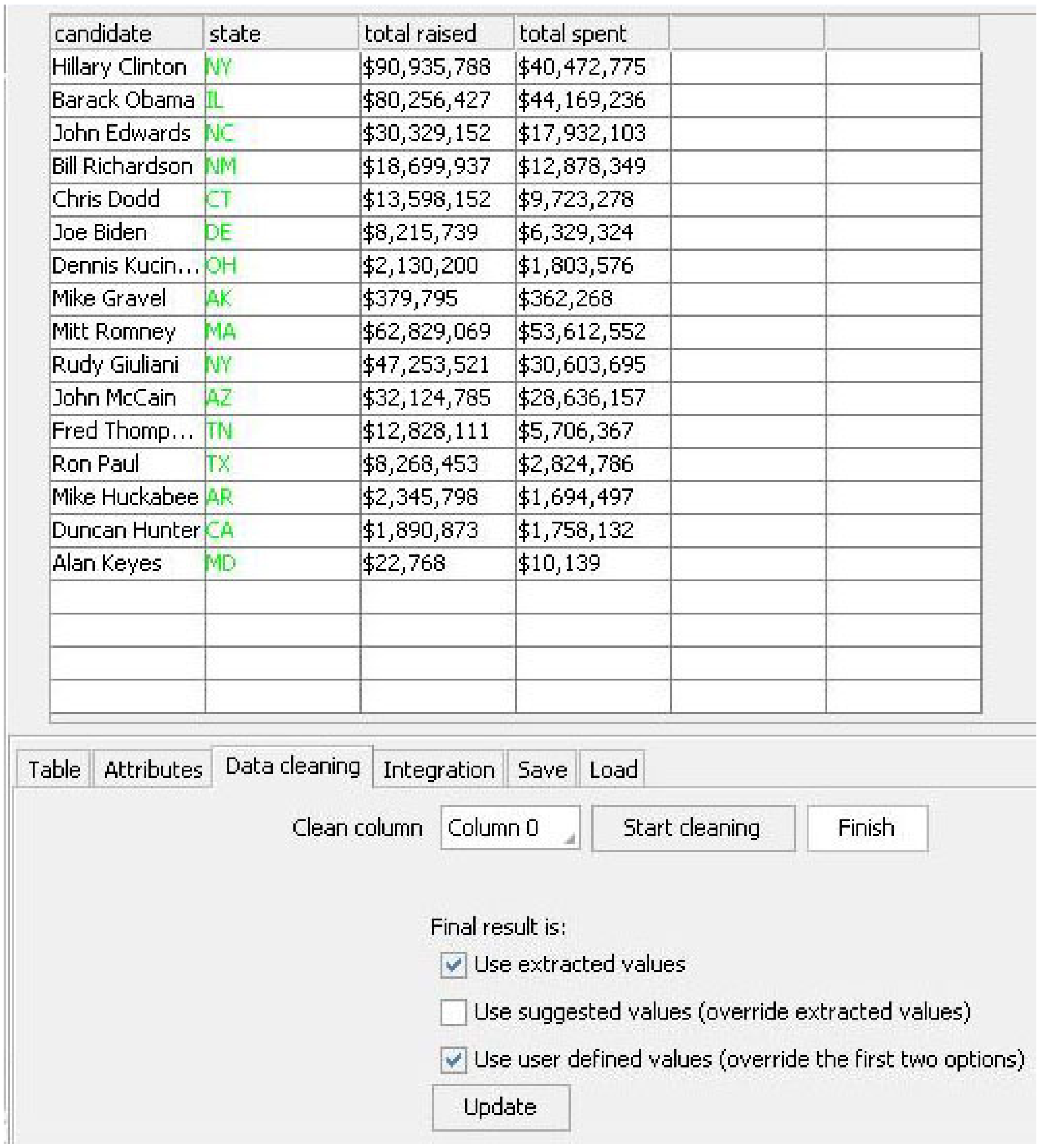
candidate	suggest	user defined	final	state	total raised	total spent
Clinton, Hillary		Hillary Clinton	Hillary Clinton	NY	$90,935,788	$40,472,775
Obama, Bar...		Barack Obama	Barack Obama	IL	$80,256,427	$44,169,236
Edwards, John		John Edwards	John Edwards	NC	$30,329,152	$17,932,103
Richardson, Bill		Bill Richardson	Bill Richardson	NM	$18,699,937	$12,878,349
Dodd, Chris		Chris Dodd	Chris Dodd	CT	$13,598,152	$9,723,278
Biden, Joe		Joe Biden	Joe Biden	DE	$8,215,739	$6,329,324
Kucinich, De...		Dennis Kucin...	Dennis Kucin...	OH	$2,130,200	$1,803,576
Gravel, Mike		Mike Gravel	Mike Gravel	AK	$379,795	$362,268
Romney, Mitt		Mitt Romney	Mitt Romney	MA	$62,829,069	$53,612,552
Giuliani, Rudy		Rudy Giuliani	Rudy Giuliani	NY	$47,253,521	$30,603,695
McCain, John		John McCain	John McCain	AZ	$32,124,785	$28,636,157
Thompson, ...		Fred Thomp...	Fred Thomp...	TN	$12,828,111	$5,706,367
Paul, Ron		Ron Paul	Ron Paul	TX	$8,268,453	$2,824,786
Huckabee, ...		Mike Huckabee	Mike Huckabee	AR	$2,345,798	$1,694,497
Hunter, Dun...		Duncan Hunter	Duncan Hunter	CA	$1,890,873	$1,758,132
Keyes, Alan		Alan Keyes	Alan Keyes	MD	$22,768	$10,139

Clean column: Column 0

Final result is:

- [✓] Use extracted values
- [ ] Use suggested values (override extracted values)
- [✓] Use user defined values (override the first two options)

Figure 5.7: The user clicks the *Update* button to update the result on the *final* column.



candidate	state	total raised	total spent		
Hillary Clinton	NY	$90,935,788	$40,472,775		
Barack Obama	IL	$80,256,427	$44,169,236		
John Edwards	NC	$30,329,152	$17,932,103		
Bill Richardson	NM	$18,699,937	$12,878,349		
Chris Dodd	CT	$13,598,152	$9,723,278		
Joe Biden	DE	$8,215,739	$6,329,324		
Dennis Kucin...	OH	$2,130,200	$1,803,576		
Mike Gravel	AK	$379,795	$362,268		
Mitt Romney	MA	$62,829,069	$53,612,552		
Rudy Giuliani	NY	$47,253,521	$30,603,695		
John McCain	AZ	$32,124,785	$28,636,157		
Fred Thomp...	TN	$12,828,111	$5,706,367		
Ron Paul	TX	$8,268,453	$2,824,786		
Mike Huckabee	AR	$2,345,798	$1,694,497		
Duncan Hunter	CA	$1,890,873	$1,758,132		
Alan Keyes	MD	$22,768	$10,139		

Clean column: Column 0

Final result is:

[✓] Use extracted values

[ ] Use suggested values (override extracted values)

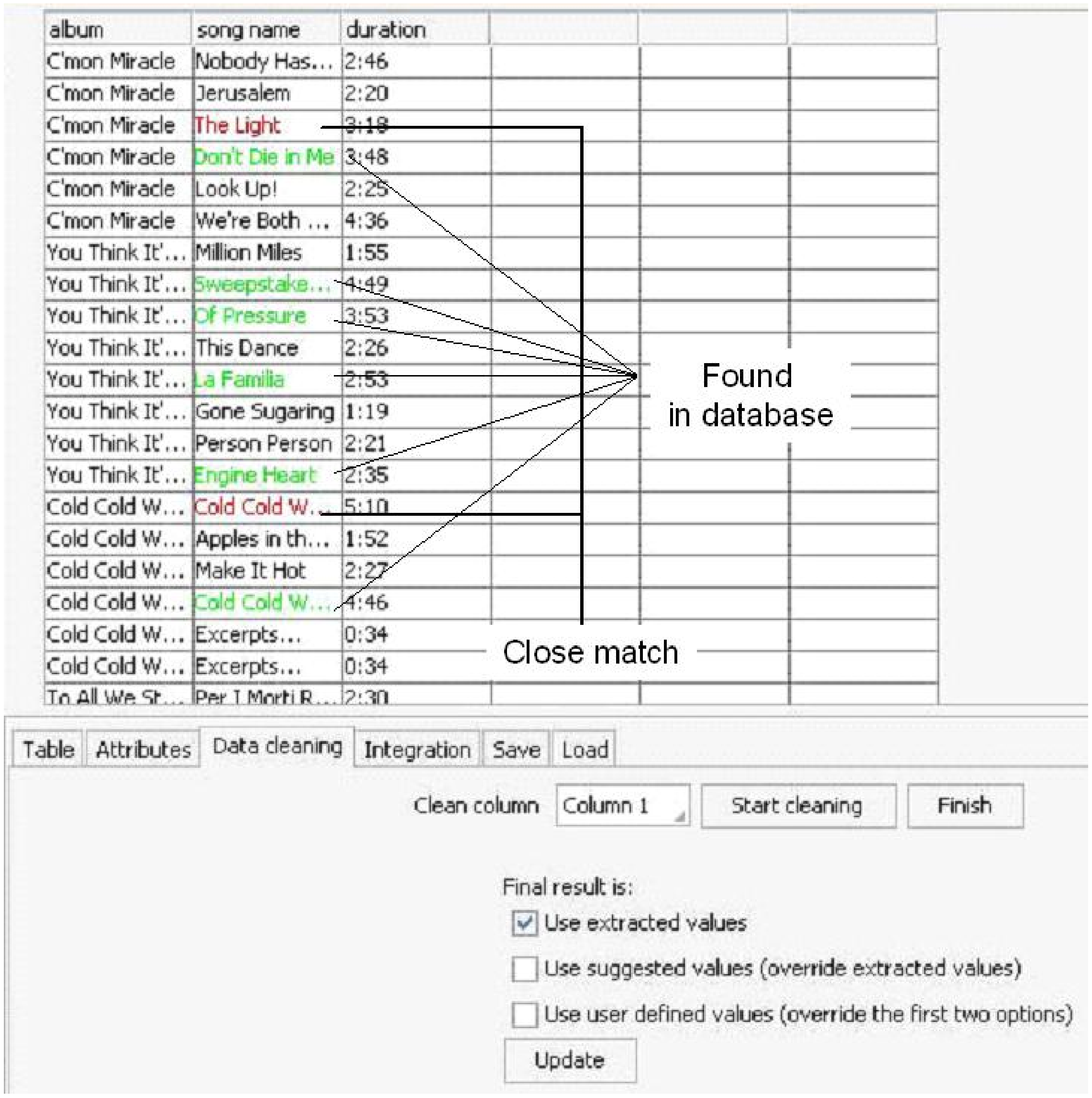
[✓] Use user defined values (override the first two options)

Figure 5.8: When the user clicks *Finish*, Karma exits the data cleaning mode and the original data is replaced with the cleaned data.

If the user wants to use the *user defined* column as the final result, the user checks the box *User user defined values (override the first two options)* as shown in Figure 5.6. There are three checkboxes. The first checkbox tells Karma to use the original value (no cleaning). The second checkbox tells Karma to use original values, but replace them with alternatives for the case where Karma have suggestion. The third checkbox tells Karma that the data in the *user defined* column will have the highest priority over the first two choices.

Once the user clicks the *Update* button, the *final* column will be updated to reflect the chosen checkboxes (Figure 5.7). If the user is not satisfied with the final result, the user can also modify the result in the *final* column by selectively typing in new values. If the user is happy with the result, the user can click the *Finish* button to exit the data cleaning mode. Figure 5.8 shows what the table looks like after the data cleaning operation; the naming format is changed from *Lastname, Firstname* to *Firstname Lastname.*

The second example, shown from Figure 5.9 through Figure 5.13, deals with the case where there are misspellings in the data. In our music example, Karma compares new data elements with existing data elements in the database. Any new element that matches perfectly with existing data elements will be shown in green. On the other hand, any new element that closely matches existing data elements will be shown in red. In Figure 5.9, *The Light* and *Cold Cold Water [Album Version]* are shown as red, while other lighter font elements are shown in green. When the user chooses *Column 1* (song name) and clicks *Start cleaning*, Karma enters the data cleaning mode. Figure 5.10 shown the replacement suggestion provided by Karma.



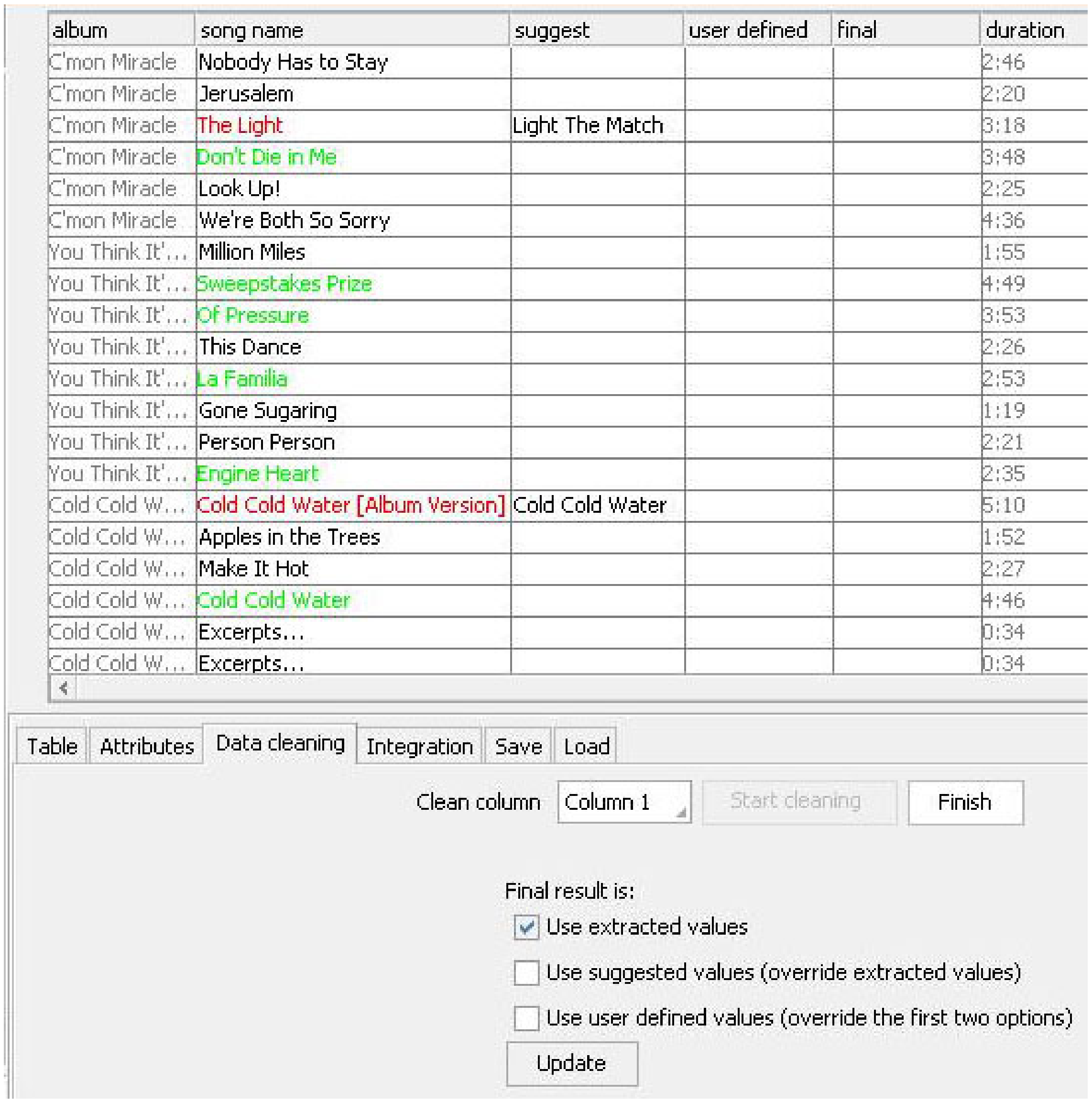
album	song name	duration
C'mon Miracle	Nobody Has...	2:46
C'mon Miracle	Jerusalem	2:20
C'mon Miracle	The Light	3:18
C'mon Miracle	Don't Die in Me	3:48
C'mon Miracle	Look Up!	2:25
C'mon Miracle	We're Both ...	4:36
You Think It'...	Million Miles	1:55
You Think It'...	Sweepstake...	4:49
You Think It'...	Of Pressure	3:53
You Think It'...	This Dance	2:26
You Think It'...	La Familia	2:53
You Think It'...	Gone Sugaring	1:19
You Think It'...	Person Person	2:21
You Think It'...	Engine Heart	2:35
Cold Cold W...	Cold Cold W...	5:10
Cold Cold W...	Apples in th...	1:52
Cold Cold W...	Make It Hot	2:27
Cold Cold W...	Cold Cold W...	4:46
Cold Cold W...	Excerpts...	0:34
Cold Cold W...	Excerpts...	0:34
To All We St...	Per I Morti R...	2:30

Clean column: Column 1

Final result is:

- [✓] Use extracted values
- [ ] Use suggested values (override extracted values)
- [ ] Use user defined values (override the first two options)

Figure 5.9: In the music example, the lighter font indicates that either a data element is found within the database (shown as green in Karma), or a close match has been found which might indicate misspelling (shown as red in Karma).



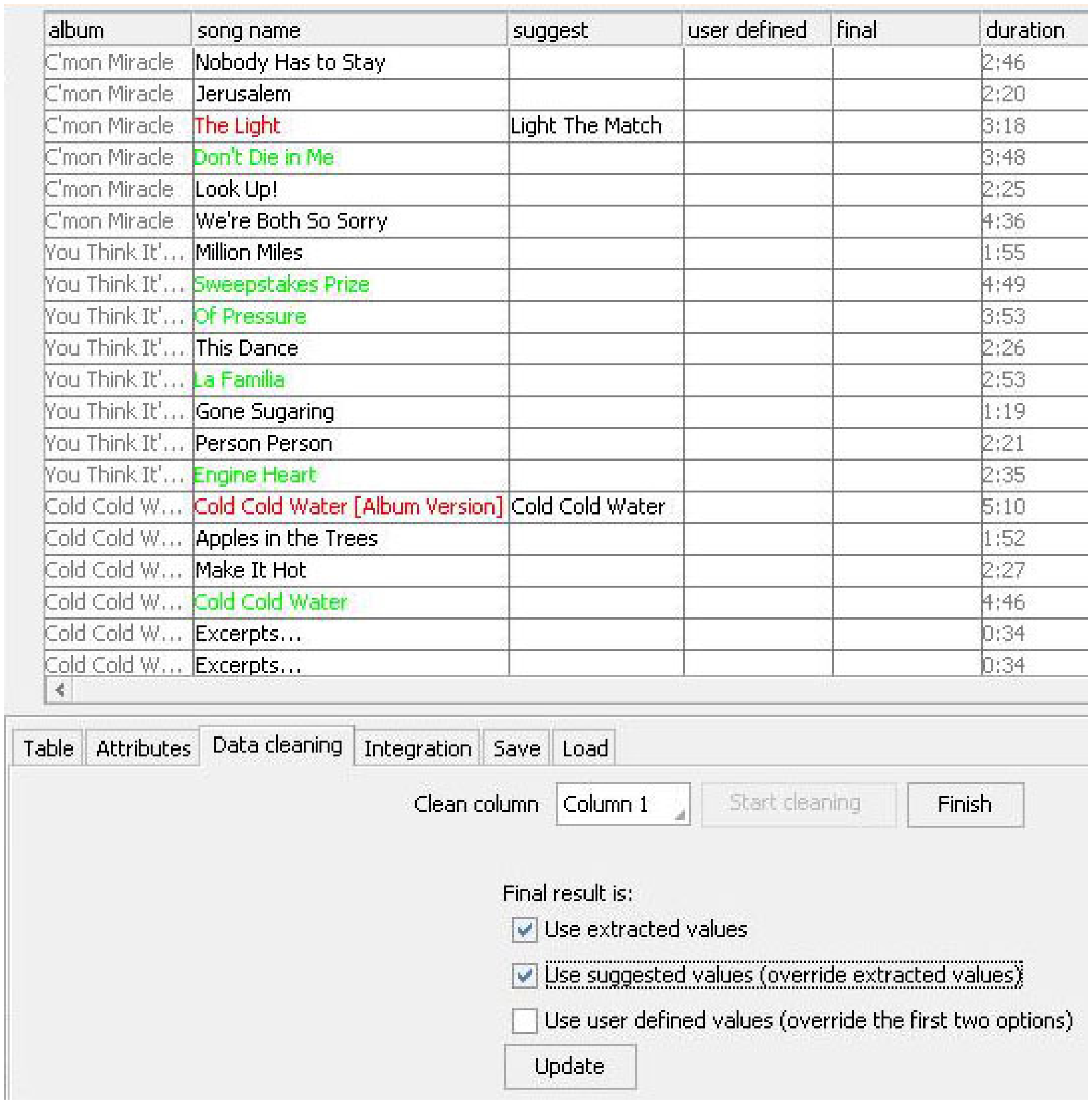
album	song name	suggest	user defined	final	duration
C'mon Miracle	Nobody Has to Stay				2:46
C'mon Miracle	Jerusalem				2:20
C'mon Miracle	The Light	Light The Match			3:18
C'mon Miracle	Don't Die in Me				3:48
C'mon Miracle	Look Up!				2:25
C'mon Miracle	We're Both So Sorry				4:36
You Think It'...	Million Miles				1:55
You Think It'...	Sweepstakes Prize				4:49
You Think It'...	Of Pressure				3:53
You Think It'...	This Dance				2:26
You Think It'...	La Familia				2:53
You Think It'...	Gone Sugaring				1:19
You Think It'...	Person Person				2:21
You Think It'...	Engine Heart				2:35
Cold Cold W...	Cold Cold Water [Album Version]	Cold Cold Water			5:10
Cold Cold W...	Apples in the Trees				1:52
Cold Cold W...	Make It Hot				2:27
Cold Cold W...	Cold Cold Water				4:46
Cold Cold W...	Excerpts...				0:34
Cold Cold W...	Excerpts...				0:34

Clean column: Column 1

Final result is:

- [✓] Use extracted values
- [ ] Use suggested values (override extracted values)
- [ ] Use user defined values (override the first two options)

Figure 5.10: When the user activates the data cleaning mode, Karma shows the suggestion for replacement when there are close matches.

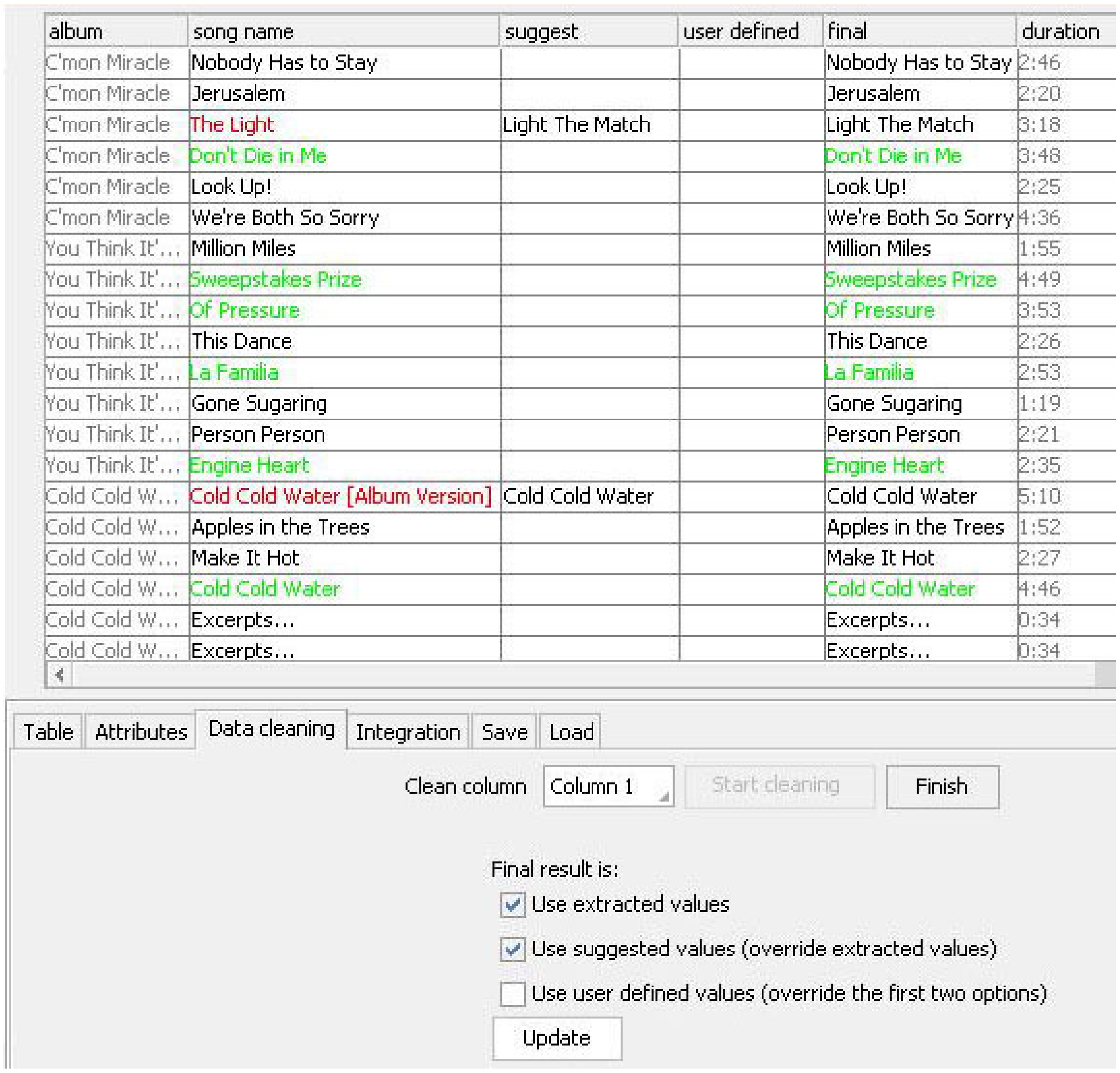


album	song name	suggest	user defined	final	duration
C'mon Miracle	Nobody Has to Stay				2:46
C'mon Miracle	Jerusalem				2:20
C'mon Miracle	The Light	Light The Match			3:18
C'mon Miracle	Don't Die in Me				3:48
C'mon Miracle	Look Up!				2:25
C'mon Miracle	We're Both So Sorry				4:36
You Think It'...	Million Miles				1:55
You Think It'...	Sweepstakes Prize				4:49
You Think It'...	Of Pressure				3:53
You Think It'...	This Dance				2:26
You Think It'...	La Familia				2:53
You Think It'...	Gone Sugaring				1:19
You Think It'...	Person Person				2:21
You Think It'...	Engine Heart				2:35
Cold Cold W...	Cold Cold Water [Album Version]	Cold Cold Water			5:10
Cold Cold W...	Apples in the Trees				1:52
Cold Cold W...	Make It Hot				2:27
Cold Cold W...	Cold Cold Water				4:46
Cold Cold W...	Excerpts...				0:34
Cold Cold W...	Excerpts...				0:34

Clean column: Column 1

Final result is:
[✓] Use extracted values
[✓] Use suggested values (override extracted values)
[ ] Use user defined values (override the first two options)

Figure 5.11: If the user wants to replace original values with suggested values, then the user checks the *Use suggested Values* box. If all boxes are checked, then suggested values will override extracted values and user defined values will override the other two previous values.

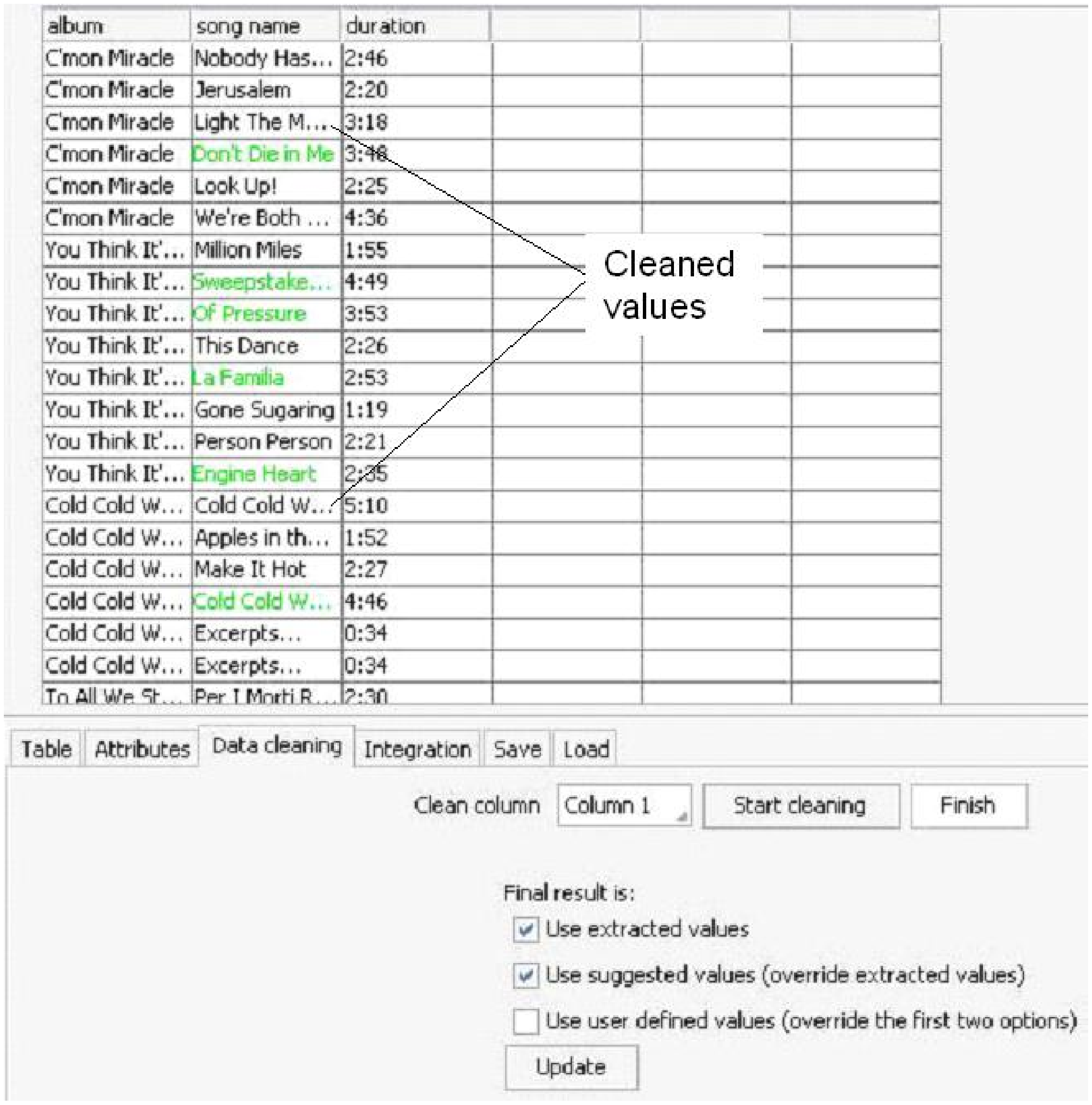


album	song name	suggest	user defined	final	duration
C'mon Miracle	Nobody Has to Stay			Nobody Has to Stay	2:46
C'mon Miracle	Jerusalem			Jerusalem	2:20
C'mon Miracle	The Light	Light The Match		Light The Match	3:18
C'mon Miracle	Don't Die in Me			Don't Die in Me	3:48
C'mon Miracle	Look Up!			Look Up!	2:25
C'mon Miracle	We're Both So Sorry			We're Both So Sorry	4:36
You Think It'...	Million Miles			Million Miles	1:55
You Think It'...	Sweepstakes Prize			Sweepstakes Prize	4:49
You Think It'...	Of Pressure			Of Pressure	3:53
You Think It'...	This Dance			This Dance	2:26
You Think It'...	La Familia			La Familia	2:53
You Think It'...	Gone Sugaring			Gone Sugaring	1:19
You Think It'...	Person Person			Person Person	2:21
You Think It'...	Engine Heart			Engine Heart	2:35
Cold Cold W...	Cold Cold Water [Album Version]	Cold Cold Water		Cold Cold Water	5:10
Cold Cold W...	Apples in the Trees			Apples in the Trees	1:52
Cold Cold W...	Make It Hot			Make It Hot	2:27
Cold Cold W...	Cold Cold Water			Cold Cold Water	4:46
Cold Cold W...	Excerpts...			Excerpts...	0:34
Cold Cold W...	Excerpts...			Excerpts...	0:34

Final result is:

- [✓] Use extracted values
- [✓] Use suggested values (override extracted values)
- [ ] Use user defined values (override the first two options)

Figure 5.12: After the user clicks the *Update* button, the *final* column reflects the chosen options. In this case, any non-empty suggest values replace the original values.



album	song name	duration			
C'mon Miracle	Nobody Has...	2:46			
C'mon Miracle	Jerusalem	2:20			
C'mon Miracle	Light The M...	3:18			
C'mon Miracle	Don't Die in Me	3:48			
C'mon Miracle	Look Up!	2:25			
C'mon Miracle	We're Both ...	4:36			
You Think It'...	Million Miles	1:55			
You Think It'...	Sweepstake...	4:49			
You Think It'...	Of Pressure	3:53			
You Think It'...	This Dance	2:26			
You Think It'...	La Familia	2:53			
You Think It'...	Gone Sugaring	1:19			
You Think It'...	Person Person	2:21			
You Think It'...	Engine Heart	2:35			
Cold Cold W...	Cold Cold W...	5:10			
Cold Cold W...	Apples in th...	1:52			
Cold Cold W...	Make It Hot	2:27			
Cold Cold W...	Cold Cold W...	4:46			
Cold Cold W...	Excerpts...	0:34			
Cold Cold W...	Excerpts...	0:34			
To All We St...	Per I Morti R...	2:30			

Clean column: Column 1

Final result is:

- [✓] Use extracted values
- [✓] Use suggested values (override extracted values)
- [ ] Use user defined values (override the first two options)

Figure 5.13: When the user clicks *Finish*, Karma exits the data cleaning mode and the original data is replaced with the cleaned data.

Note that these suggested values do not always mean misspellings occur; it just means that there are closely matched values in the database and these values are presented to the user for consideration. Assuming that the user wants to use the suggested values, the user chooses the second checkboxes "Use suggested values (override extracted value)" (Figure 5.11) to override original values with suggested values. Once the user clicks the *Update* button, the result is update in the *final* column (Figure 5.12). If the user is satisfied with the result, the user can click the *Finish* button to exit the data cleaning mode. Figure 5.13 shows the updated data with suggested values replacing original values.

5.2.2 Transformation by Examples

In our political campaign donation example, the user specifies how to clean the data by typing in the desired result. Figure 5.14 shows a schematic diagram of how Karma processes a user defined input. The original data element *Clinton, Hillary* is sent to a set of predefined transformations to generate the output. If the output generated by a transformation equals to the user defined input (i.e., *Hillary Clinton*), then that transformation rule is selected. The selected transformation rule is then used to apply to the rest of the data in the original column (e.g., Obama, Barack).

By allowing the user to specify the end result, the user does not need to know how to specify the transformation rule, which can be challenging. For example, the regex widget in Figure 1.3 specifies the regular expression (i.e., replace (\w*\s)(\w*) with $2 $1) that transform "*Lastname Firstname*" to "*Firstname Lastname*". While this approach does not require a user to understand programming, it does have three limitations:

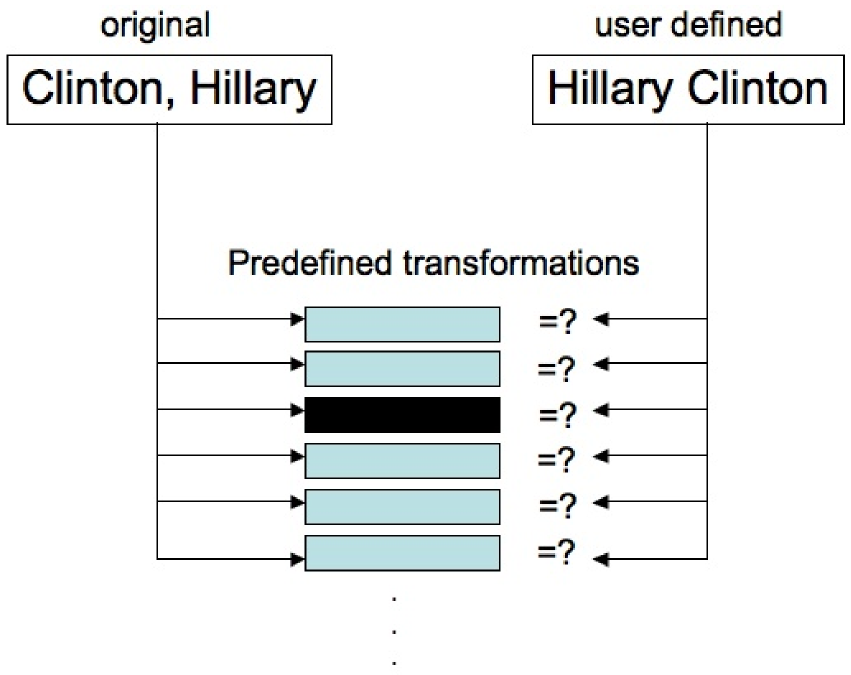


Figure 5.14: A diagram showing how Karma processes a user defined cleaning input. When the user specifies an example, the original data and the user defined data are sent to a set of predefined transformation rules to determine which rule can be used.

Table 5.1: Predefined transformation supported by Karma

Name	Function
Symbol Substitution	Allow a single symbol replacement i.e., 12-30-2008 → 12/30/2008
Name Reverse	Lastname, Firstname i.e., Obama, Barack → Barack Obama
Name Reverse2	Firstname, Lastname i.e., Barack Obama → Obama, Barack
All Uppercase	Change every character to uppercase i.e., Hillary → HILLARY
All Lowercase	Change every character to lowercase i.e., Hillary → hillary
Word Capital	Capitalize the first character of each token i.e., barack obama → Barack Obama
Substitution	Allow a single token replacement i.e., 3 Ames St. → 3 Ames Street
SubstringEqLength	Similar to java substring i.e., 28 reviews → 28

- *There may be no predefined transformation match*: The user would need to type in the cleaned values manually. Table 5.1 lists available transformations supported by Karma.

- *More than one rule can be matched*: Currently, Karma selects the first rule that matches. In the future work, the user can be prompted with an option to select one rule from matching rules.

- *More than one transformations are needed to clean a data column*: There maybe some cases where a data column contains data in a mixed format. For example, the first row might contain the data "*12/03/2008*", while the second row might contain the data "*March 15, 2006*". To clean a column of mixed format data, more than one transformation rule is required. However, the current version of Karma, while it can be extended to support multiple transformation rules, only supports one transformation rule for each column.

5.2.3 Discrepancy Detection: Misspelling

In our music example, Karma hints to the user that some data elements might need to be cleaned by applying red on them. When the user enters the data cleaning mode, she is presented with a possible replacement for each red data element. The replacement is computed using the following algorithm:

Let:

1. A = the attribute of new data column.

2. V_A = the value set under the attribute A.

3. S = set of data sources in the database where each source has the attribute A as one of their column.

4. V_S = the union of all the values under the attribute A from each source in S.

5. F(v1,v2) = a string similarity function that takes in two data elements and return the number ranging from 0 to 1. The closer the result to 1, the more similar between v1 and v2.

6. X = the suggestion list for an arbitrary value v in V_A.

Then, the suggestion list for each value in V_A is computed using the following formula: X = $\{ v_s | v_a \in V_A \wedge v_s \in V_S \wedge \mathrm{F}(v_s,v_a) > 0.75\}$. Specifically, X is computed by comparing each value in V_A with each value in V_S using a string similarity algorithm. In Karma, we choose unsmoothed Jacard similiarty [15] for its fast computation and performance. If the two strings are similar enough, the value in V_S is added to X. For each value in V_A where its corresponding X is not empty, Karma sets the color of that data element to be red on the table. When the user chooses to clean the data, Karma presents the user with suggestions for replacement for each red element. If there is more than one suggestion, the user can click on the suggestion cell to see the drop down list of available suggestions.

The performance of this approach is tied to how well the source modeling problem is addressed. If the user selects a wrong attribute for a newly extracted data column, then Karma will not be able to use the correct set of data to compute the suggestion set for replacement. However, even if the user selects the wrong attribute, Karma does no worse than other Mashup tools as they do not provide any support to detect misspellings in the extracted data.

5.3 Discussion

The data cleaning approach in this chapter fits the Karma framework by using all three key ideas mentioned in chapter 1. First, the user focuses on the data. By specifying the cleaning result that the user wants to see, she indirectly induces the cleaning transformation function. Second, Karma leverages existing databases to detect misspelling, so the user does not have to spend time locating these errors. Third, as soon as the source modeling problem is solved (e.g., attribute is specified), Karma uses the information about that attribute to help determine which existing data set should be used to detect misspelling.

Chapter 6

Data Integration

This chapter is divided into four sections. The first section defines what data integration is and why it is important. The second section shows two scenarios of how a user would interact with Karma to solve the data integration problem. The third section explains the technical details in the backend that enable such interactions. The fourth section elaborates on how my approach to the data integration problem fits the overall Karma framework.

6.1 Problem Definition

From chapter 3 through chapter 5, we have focused on instances where the user retrieves, models, and cleans data from one single source. The data integration problem takes that source and figures out how to combine that data source with other data sources. For example, the political data source used in the two previous chapters is interesting, but it lacks party affiliation or candidate ages. If we can combine the political data source with another data source that contains such information, the overall data will be more complete and more meaningful. There are two overall issues that we need to examine.

The first issue is locating and obtaining related data sources. The second issue is figuring out how to combine data sources.

When building Mashups, if a user knows which data from which sites to combine, then the first issue is solved. However, given the magnitude of existing web pages, locating good related sources maybe difficult and time consuming.

Assuming that related sources can be located, the next step is to figure out how to combine them. This issue can be thought of as a query building problem where we need to come up with a query (i.e., SQL) that combines multiple data tables together. However, such queries can be complex due to the composition of data sources:

- *Binding sources*: Some sources that have form input, such as the tea example in chapter 3, has binding constraints. To get the data, we need to supply input values.
- *Multiple sources coverage*: Sometimes more than two sources are required during the data integration process. For example, to retrieve the age of each presidential candidate, we might need to access two sources: one with age information for candidates belonging to the Democrat party, and another source with age information for candidates belonging to the Republican party.
- *Multiple linking*: Sometimes multiple sources must be linked together in order to access the needed data. For example, the Google music example contains only album information, but not reviews. To obtain review ratings, we might need to link the Google music data source with the Amazon data source using a database join over artist and album name to retrieve the review rating.

The next section will show the case study walkthrough that lets the user combine multiple data sources; and the approach section will elaborate on how Karma solves the two issues above.

6.2 Case Study Walkthrough

There are two examples in this section. The first example continues to expand from the political contribution scenario, while the second example is a new example showing how a user can build a table of data derived from database without having to extract data from a new source.

6.2.1 Mashup Building

Given the political campaign data, the user can ask Karma to suggest what related information she can get. This information comes from existing databases. These databases grow larger as users extract more data from new sources and save them into databases. The user can start the data integration process by going into the *Integration* tab as shown in Figure 6.1. Once the data integration mode is activated, the user can get suggestions of new attributes that could be added to the table. This is done by going into the *Attributes* tab and selecting one of the empty columns (e.g., Column 4) as shown in Figure 6.2.

Once the user selects the new column to work on, Karma will show what kinds of information it can get from the existing database. Figure 6.3 shows three available choices: age, full_state, and party. This means that based on the data from the political campaign source, Karma can get candidate's age information, the full name for each state, and the party affiliation. Let us assume that the user selects *age* (Figure 6.4) and clicks the *Update* button (Figure 6.5), the fifth column on the table will be updated with the new attribute information. The user can repeat the process by continuing to add more attributes in empty columns as shown in Figure 6.6 and Figure 6.7.

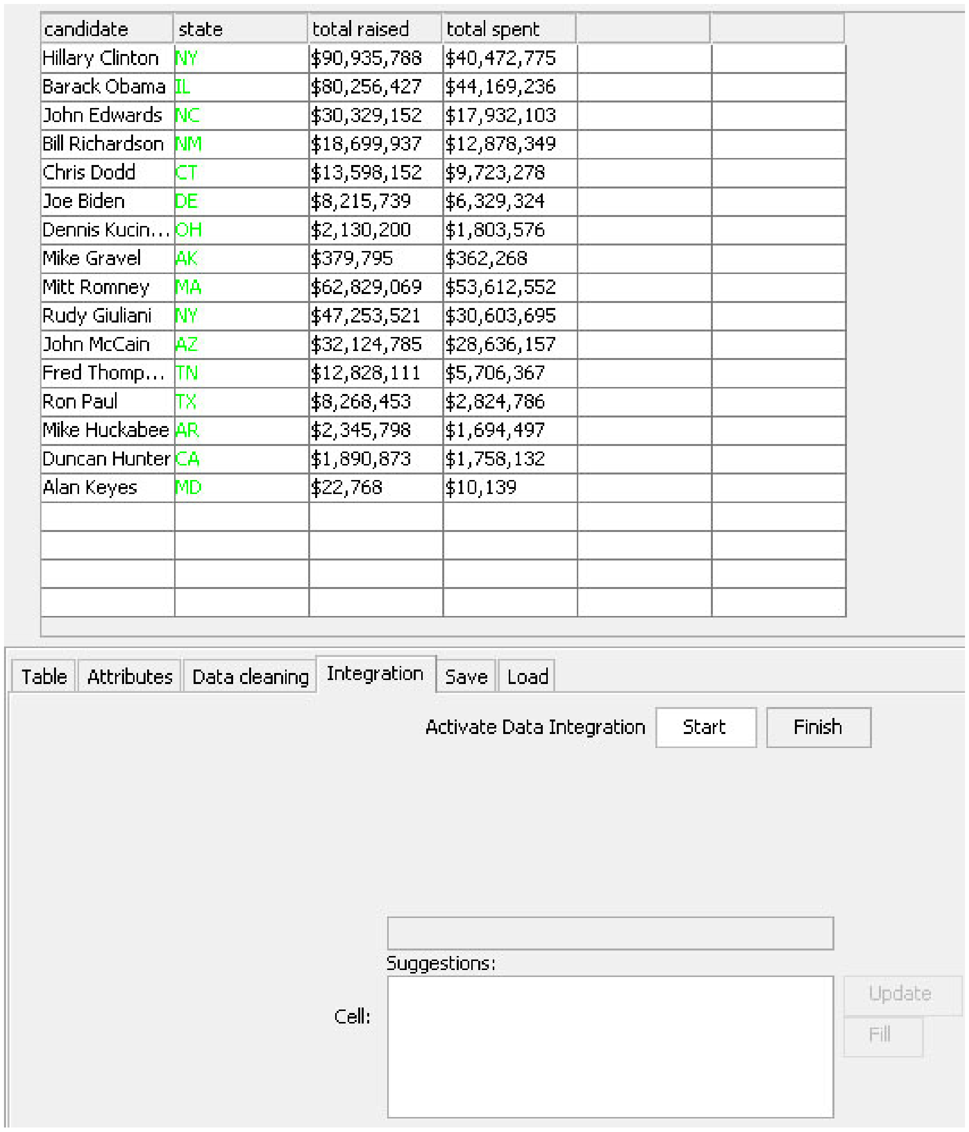
After the user finishes adding new attributes, the user can click the button *Fill* under the *Attributes* tab. Karma will then automatically generate the queries that combine the new source (i.e., political campaign) with existing sources from databases so the table is filled as shown in Figure 6.8. The user is sheltered from the detail of sources and query formulations and is presented with information that Karma can integrate.

While focuses of the thesis is not on data display, if the table contains any geospatial information (e.g., State), the user can display the data on the map by clicking on the *Map* button in the *Table* tab. Figure 6.9 show the data in the table displayed on a Google map. The state information is used as a location for each data point. When the user clicks on a point, the corresponding data from the table will be shown.

6.2.2 Building Query by Example

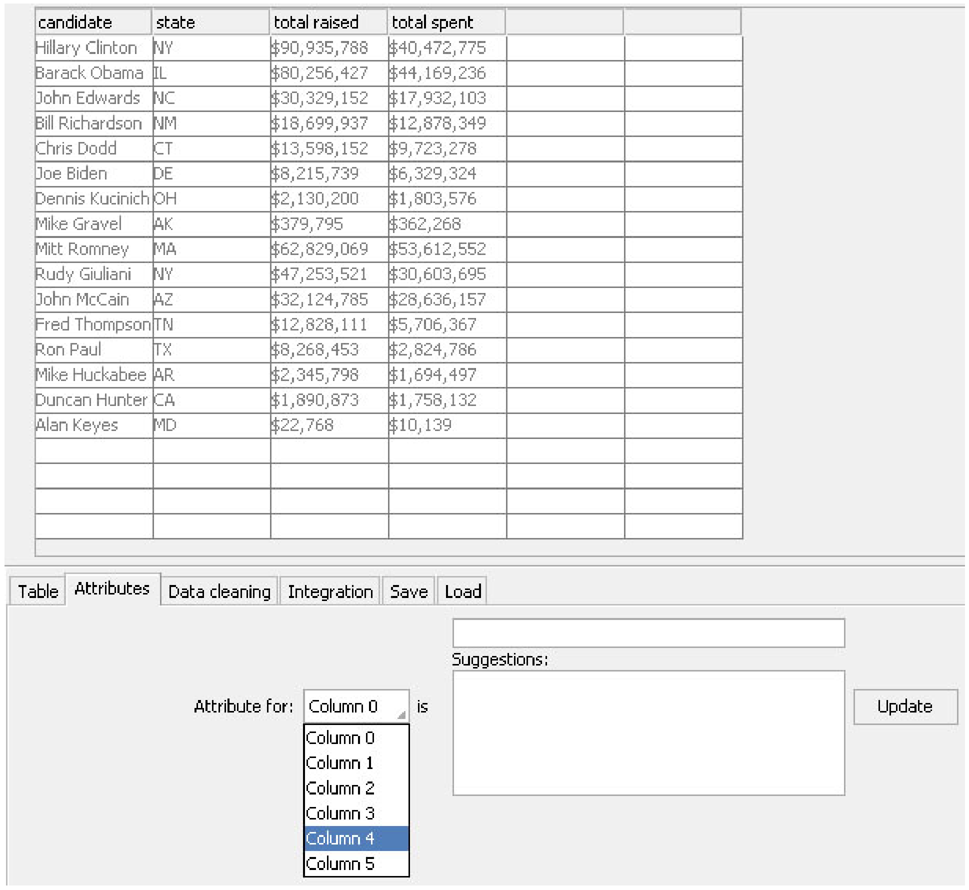
Suppose we want to find good Japanese and Vietnamese restaurants in Los Angeles. TIn addition, these restaurants must have an **A** health rating. Because restaurant review sites do not contain the health rating information, combining data from multiple sources is necessary. With these criteria in mind, we let the user create a table by interactively filling in values using Karma's data integration mode. Figure 6.10 shows what the end result should look like.

To create the data table shown in Figure 6.10, the user can enter values in a method similar to Google Suggest (http://labs.google.com/suggest); Google Suggest suggests a



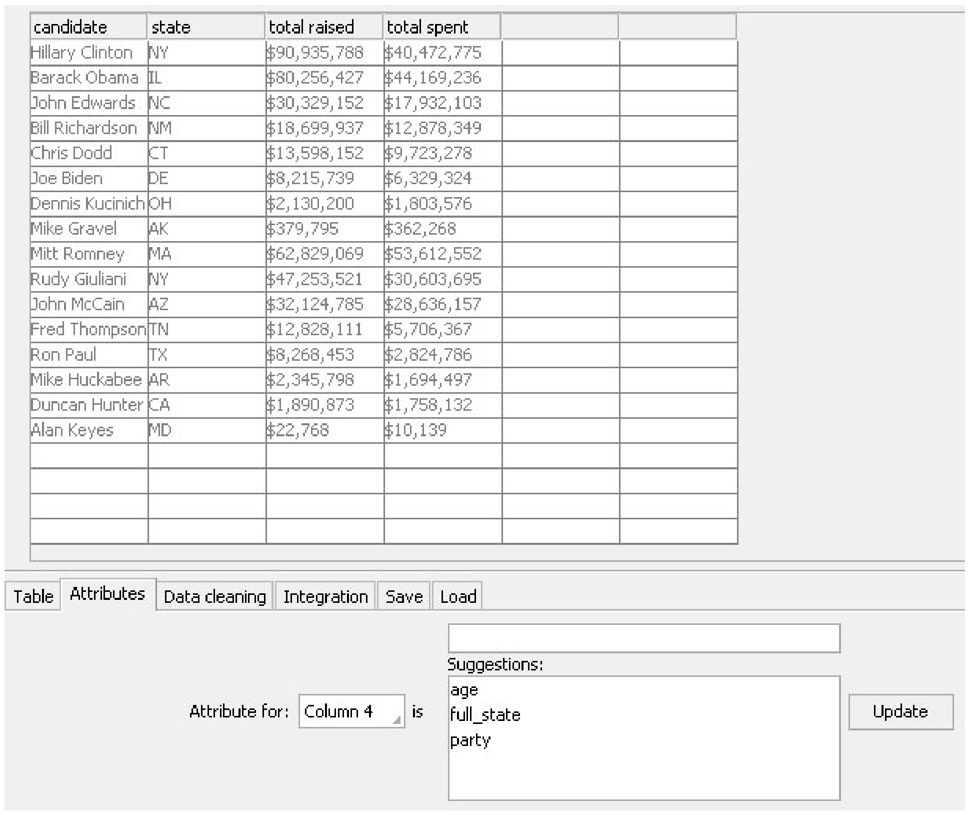
candidate	state	total raised	total spent		
Hillary Clinton	NY	$90,935,788	$40,472,775		
Barack Obama	IL	$80,256,427	$44,169,236		
John Edwards	NC	$30,329,152	$17,932,103		
Bill Richardson	NM	$18,699,937	$12,878,349		
Chris Dodd	CT	$13,598,152	$9,723,278		
Joe Biden	DE	$8,215,739	$6,329,324		
Dennis Kucin...	OH	$2,130,200	$1,803,576		
Mike Gravel	AK	$379,795	$362,268		
Mitt Romney	MA	$62,829,069	$53,612,552		
Rudy Giuliani	NY	$47,253,521	$30,603,695		
John McCain	AZ	$32,124,785	$28,636,157		
Fred Thomp...	TN	$12,828,111	$5,706,367		
Ron Paul	TX	$8,268,453	$2,824,786		
Mike Huckabee	AR	$2,345,798	$1,694,497		
Duncan Hunter	CA	$1,890,873	$1,758,132		
Alan Keyes	MD	$22,768	$10,139		

Figure 6.1: The user begins the process of data integration by choosing the *Integration* tab and clicks *Start*.



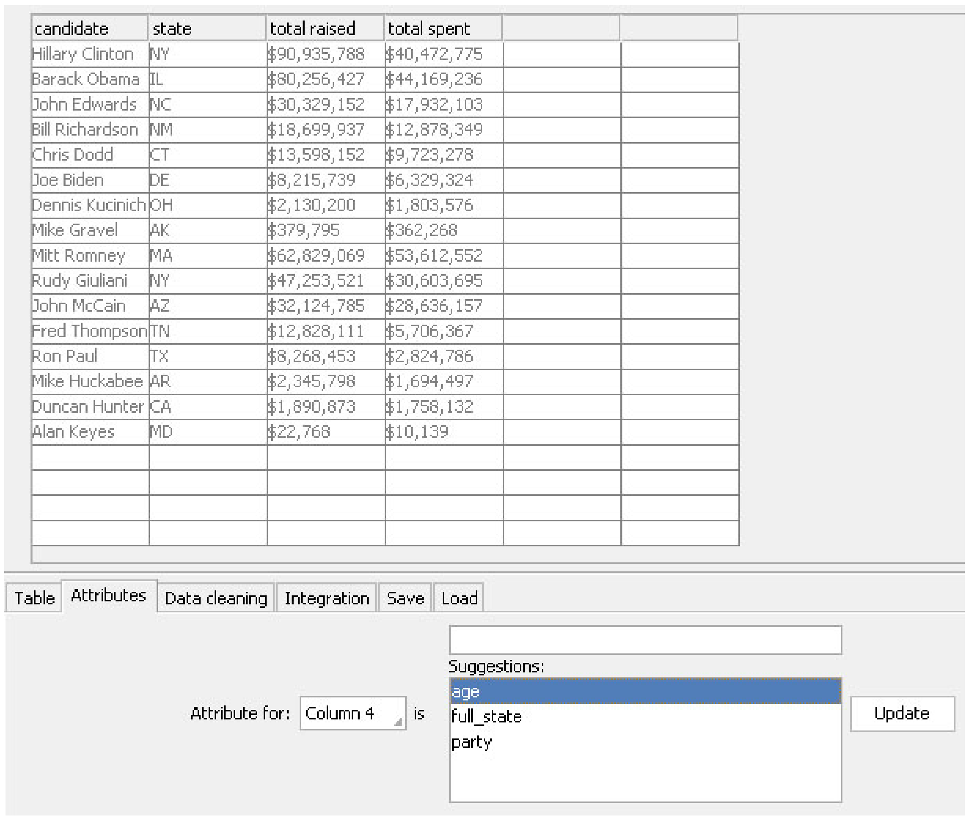
candidate	state	total raised	total spent		
Hillary Clinton	NY	$90,935,788	$40,472,775		
Barack Obama	IL	$80,256,427	$44,169,236		
John Edwards	NC	$30,329,152	$17,932,103		
Bill Richardson	NM	$18,699,937	$12,878,349		
Chris Dodd	CT	$13,598,152	$9,723,278		
Joe Biden	DE	$8,215,739	$6,329,324		
Dennis Kucinich	OH	$2,130,200	$1,803,576		
Mike Gravel	AK	$379,795	$362,268		
Mitt Romney	MA	$62,829,069	$53,612,552		
Rudy Giuliani	NY	$47,253,521	$30,603,695		
John McCain	AZ	$32,124,785	$28,636,157		
Fred Thompson	TN	$12,828,111	$5,706,367		
Ron Paul	TX	$8,268,453	$2,824,786		
Mike Huckabee	AR	$2,345,798	$1,694,497		
Duncan Hunter	CA	$1,890,873	$1,758,132		
Alan Keyes	MD	$22,768	$10,139		

Figure 6.2: To get the suggestion from Karma for extra attributes that can be added to an empty column, the user chooses the *Attributes* tab and chooses one of the empty columns (e.g., column 4).



candidate	state	total raised	total spent		
Hillary Clinton	NY	$90,935,788	$40,472,775		
Barack Obama	IL	$80,256,427	$44,169,236		
John Edwards	NC	$30,329,152	$17,932,103		
Bill Richardson	NM	$18,699,937	$12,878,349		
Chris Dodd	CT	$13,598,152	$9,723,278		
Joe Biden	DE	$8,215,739	$6,329,324		
Dennis Kucinich	OH	$2,130,200	$1,803,576		
Mike Gravel	AK	$379,795	$362,268		
Mitt Romney	MA	$62,829,069	$53,612,552		
Rudy Giuliani	NY	$47,253,521	$30,603,695		
John McCain	AZ	$32,124,785	$28,636,157		
Fred Thompson	TN	$12,828,111	$5,706,367		
Ron Paul	TX	$8,268,453	$2,824,786		
Mike Huckabee	AR	$2,345,798	$1,694,497		
Duncan Hunter	CA	$1,890,873	$1,758,132		
Alan Keyes	MD	$22,768	$10,139		

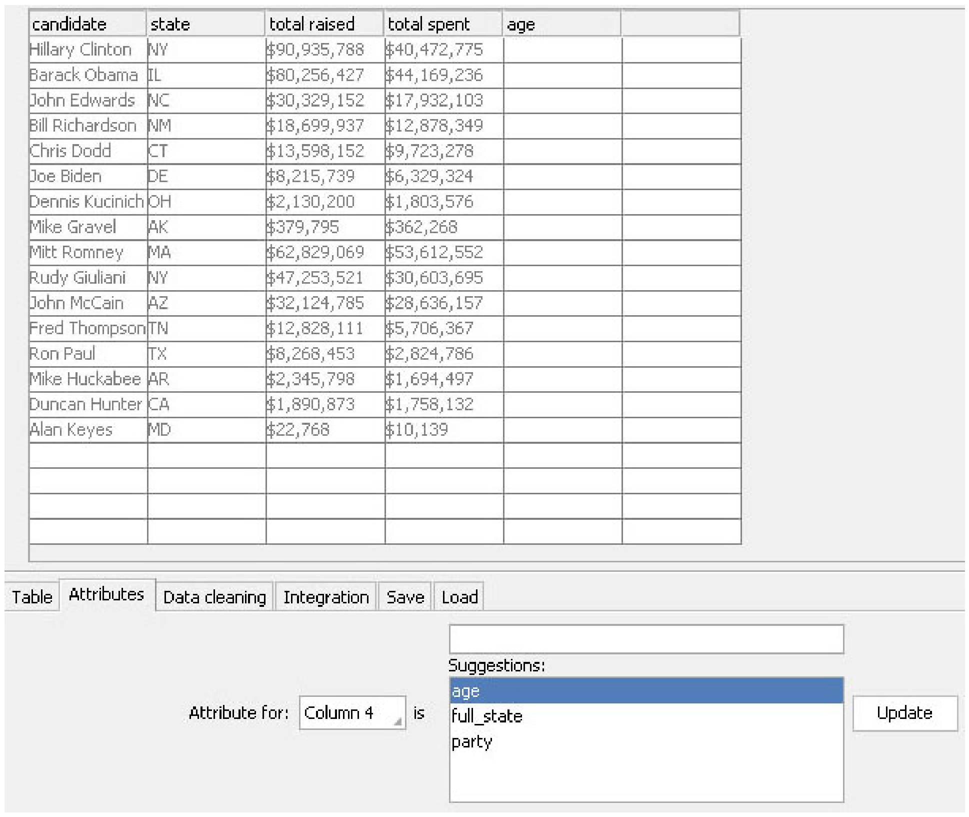
Figure 6.3: Karma suggests attributes from database that can be added to the table to expand the extracted data.



candidate	state	total raised	total spent		
Hillary Clinton	NY	$90,935,788	$40,472,775		
Barack Obama	IL	$80,256,427	$44,169,236		
John Edwards	NC	$30,329,152	$17,932,103		
Bill Richardson	NM	$18,699,937	$12,878,349		
Chris Dodd	CT	$13,598,152	$9,723,278		
Joe Biden	DE	$8,215,739	$6,329,324		
Dennis Kucinich	OH	$2,130,200	$1,803,576		
Mike Gravel	AK	$379,795	$362,268		
Mitt Romney	MA	$62,829,069	$53,612,552		
Rudy Giuliani	NY	$47,253,521	$30,603,695		
John McCain	AZ	$32,124,785	$28,636,157		
Fred Thompson	TN	$12,828,111	$5,706,367		
Ron Paul	TX	$8,268,453	$2,824,786		
Mike Huckabee	AR	$2,345,798	$1,694,497		
Duncan Hunter	CA	$1,890,873	$1,758,132		
Alan Keyes	MD	$22,768	$10,139		

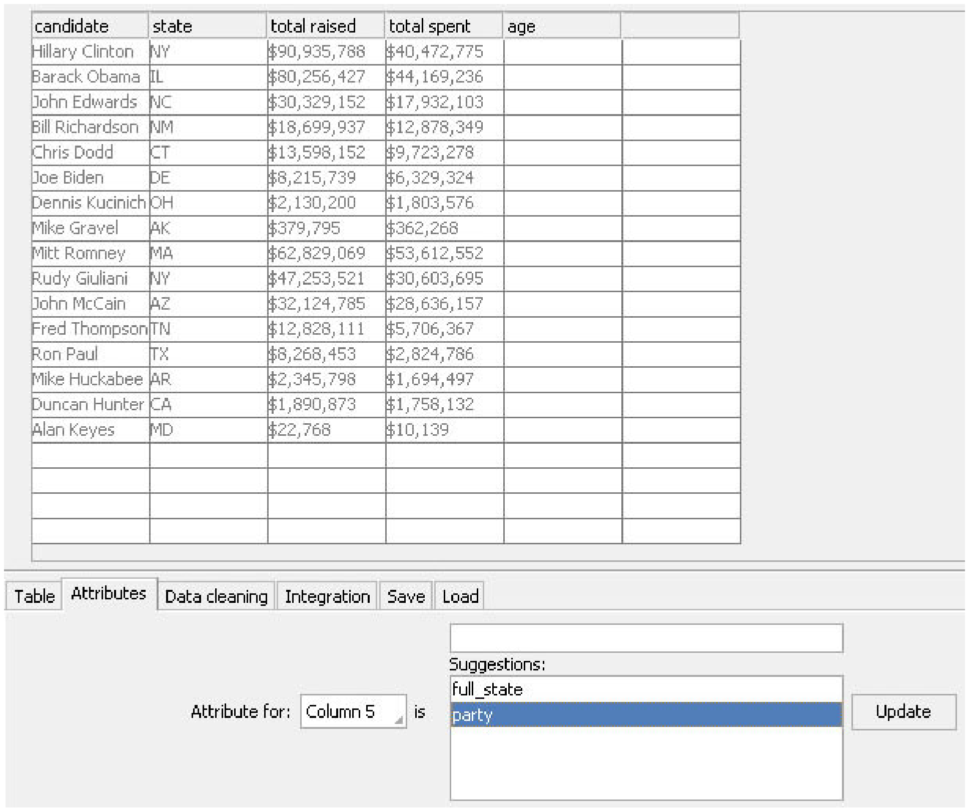
Attribute for: Column 4 is

Figure 6.4: The user can choose from the list. In this case, the user chooses *age*.



candidate	state	total raised	total spent	age	
Hillary Clinton	NY	$90,935,788	$40,472,775		
Barack Obama	IL	$80,256,427	$44,169,236		
John Edwards	NC	$30,329,152	$17,932,103		
Bill Richardson	NM	$18,699,937	$12,878,349		
Chris Dodd	CT	$13,598,152	$9,723,278		
Joe Biden	DE	$8,215,739	$6,329,324		
Dennis Kucinich	OH	$2,130,200	$1,803,576		
Mike Gravel	AK	$379,795	$362,268		
Mitt Romney	MA	$62,829,069	$53,612,552		
Rudy Giuliani	NY	$47,253,521	$30,603,695		
John McCain	AZ	$32,124,785	$28,636,157		
Fred Thompson	TN	$12,828,111	$5,706,367		
Ron Paul	TX	$8,268,453	$2,824,786		
Mike Huckabee	AR	$2,345,798	$1,694,497		
Duncan Hunter	CA	$1,890,873	$1,758,132		
Alan Keyes	MD	$22,768	$10,139		

Figure 6.5: Once the user clicks the *Update* button, the attribute on the fourth column is set to *age*.

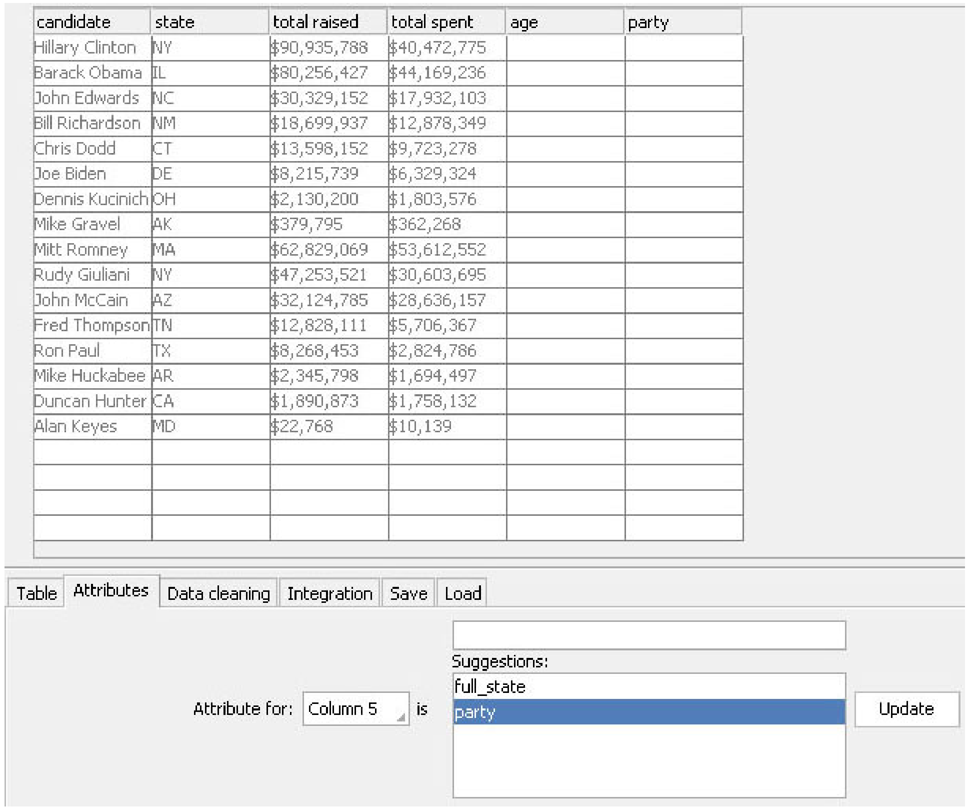


candidate	state	total raised	total spent	age	
Hillary Clinton	NY	$90,935,788	$40,472,775		
Barack Obama	IL	$80,256,427	$44,169,236		
John Edwards	NC	$30,329,152	$17,932,103		
Bill Richardson	NM	$18,699,937	$12,878,349		
Chris Dodd	CT	$13,598,152	$9,723,278		
Joe Biden	DE	$8,215,739	$6,329,324		
Dennis Kucinich	OH	$2,130,200	$1,803,576		
Mike Gravel	AK	$379,795	$362,268		
Mitt Romney	MA	$62,829,069	$53,612,552		
Rudy Giuliani	NY	$47,253,521	$30,603,695		
John McCain	AZ	$32,124,785	$28,636,157		
Fred Thompson	TN	$12,828,111	$5,706,367		
Ron Paul	TX	$8,268,453	$2,824,786		
Mike Huckabee	AR	$2,345,798	$1,694,497		
Duncan Hunter	CA	$1,890,873	$1,758,132		
Alan Keyes	MD	$22,768	$10,139		

Attribute for: Column 5 is

Suggestions:
full_state
party

Figure 6.6: The user chooses to add more attributes by selecting an empty column (i.e., Column 5) and chooses *Party*.

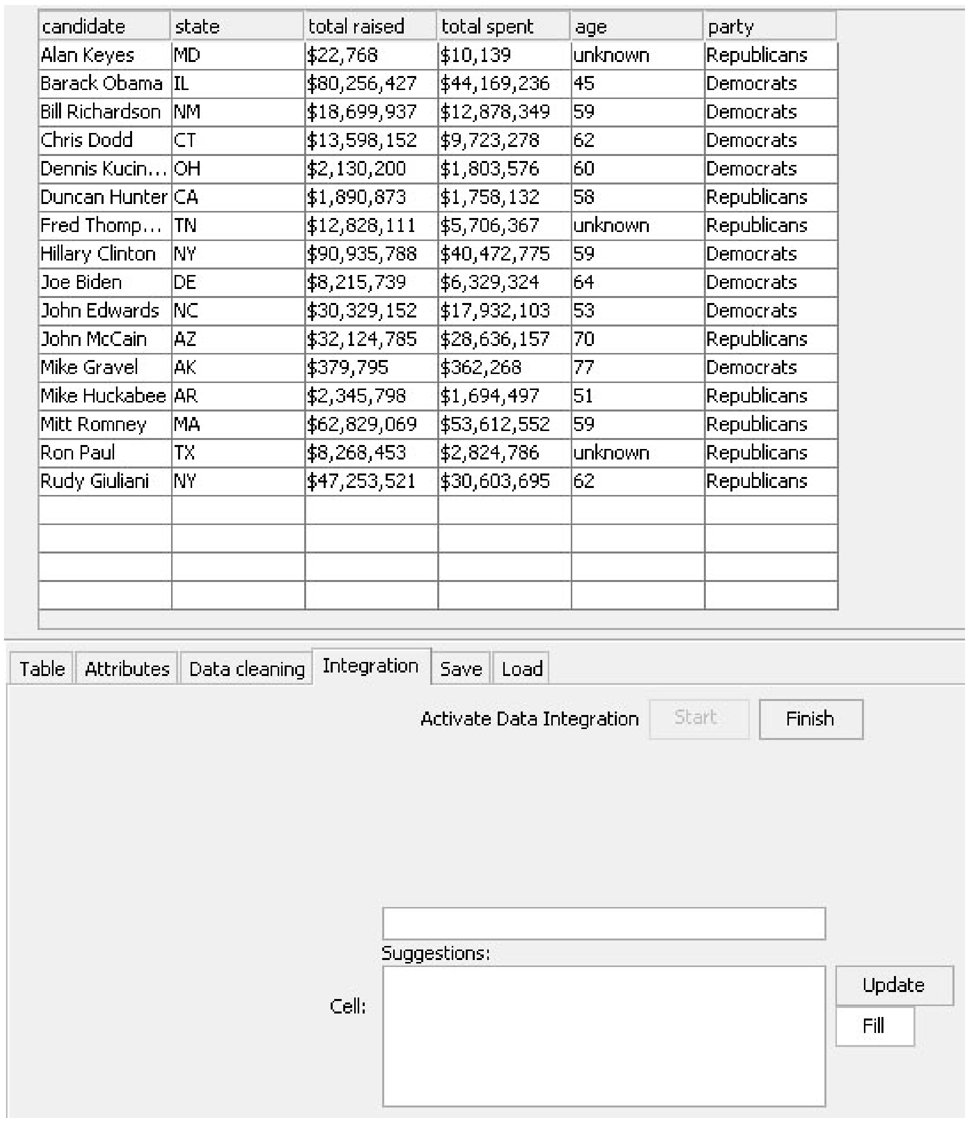


candidate	state	total raised	total spent	age	party
Hillary Clinton	NY	$90,935,788	$40,472,775		
Barack Obama	IL	$80,256,427	$44,169,236		
John Edwards	NC	$30,329,152	$17,932,103		
Bill Richardson	NM	$18,699,937	$12,878,349		
Chris Dodd	CT	$13,598,152	$9,723,278		
Joe Biden	DE	$8,215,739	$6,329,324		
Dennis Kucinich	OH	$2,130,200	$1,803,576		
Mike Gravel	AK	$379,795	$362,268		
Mitt Romney	MA	$62,829,069	$53,612,552		
Rudy Giuliani	NY	$47,253,521	$30,603,695		
John McCain	AZ	$32,124,785	$28,636,157		
Fred Thompson	TN	$12,828,111	$5,706,367		
Ron Paul	TX	$8,268,453	$2,824,786		
Mike Huckabee	AR	$2,345,798	$1,694,497		
Duncan Hunter	CA	$1,890,873	$1,758,132		
Alan Keyes	MD	$22,768	$10,139		

Attribute for: Column 5 is

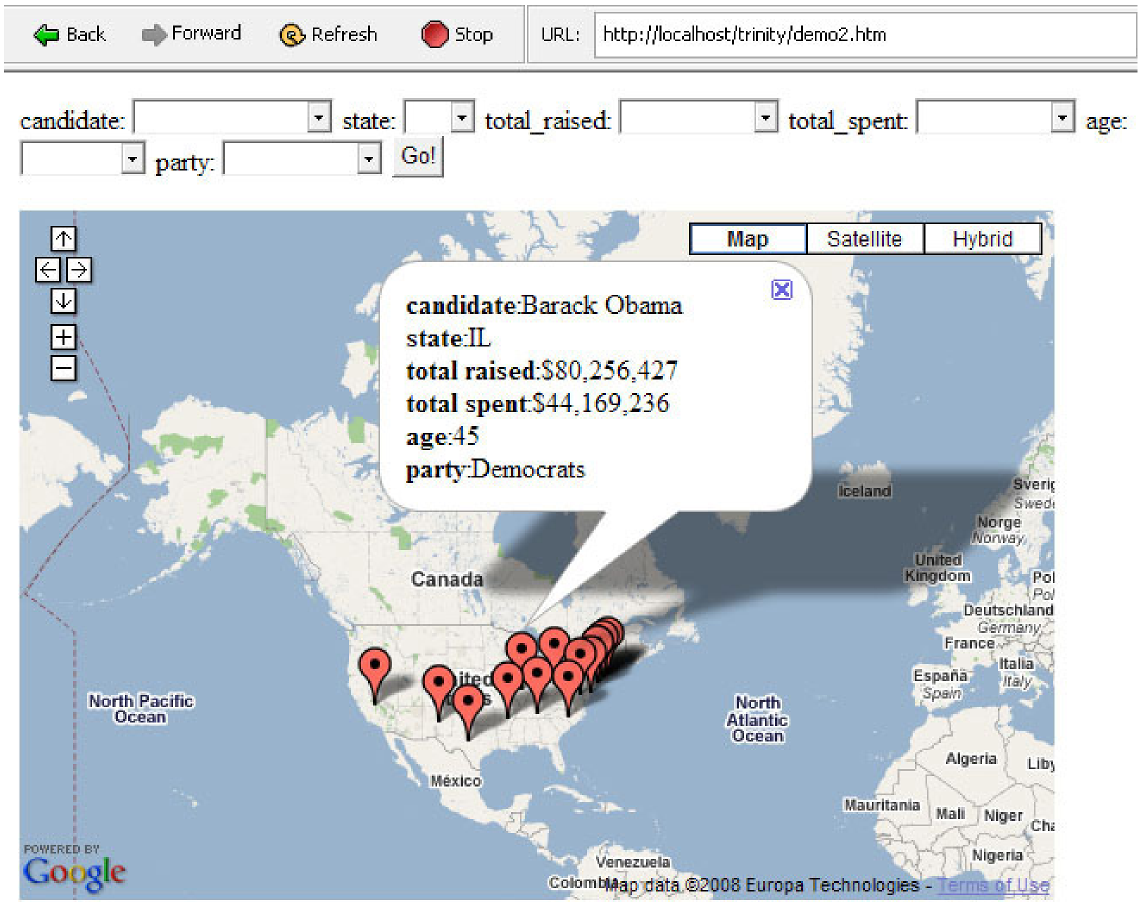
Suggestions:
full_state
party

Figure 6.7: When the user clicks the *Update* button, the fifth column attribute is changed to *Party*.



candidate	state	total raised	total spent	age	party
Alan Keyes	MD	$22,768	$10,139	unknown	Republicans
Barack Obama	IL	$80,256,427	$44,169,236	45	Democrats
Bill Richardson	NM	$18,699,937	$12,878,349	59	Democrats
Chris Dodd	CT	$13,598,152	$9,723,278	62	Democrats
Dennis Kucin...	OH	$2,130,200	$1,803,576	60	Democrats
Duncan Hunter	CA	$1,890,873	$1,758,132	58	Republicans
Fred Thomp...	TN	$12,828,111	$5,706,367	unknown	Republicans
Hillary Clinton	NY	$90,935,788	$40,472,775	59	Democrats
Joe Biden	DE	$8,215,739	$6,329,324	64	Democrats
John Edwards	NC	$30,329,152	$17,932,103	53	Democrats
John McCain	AZ	$32,124,785	$28,636,157	70	Republicans
Mike Gravel	AK	$379,795	$362,268	77	Democrats
Mike Huckabee	AR	$2,345,798	$1,694,497	51	Republicans
Mitt Romney	MA	$62,829,069	$53,612,552	59	Republicans
Ron Paul	TX	$8,268,453	$2,824,786	unknown	Republicans
Rudy Giuliani	NY	$47,253,521	$30,603,695	62	Republicans

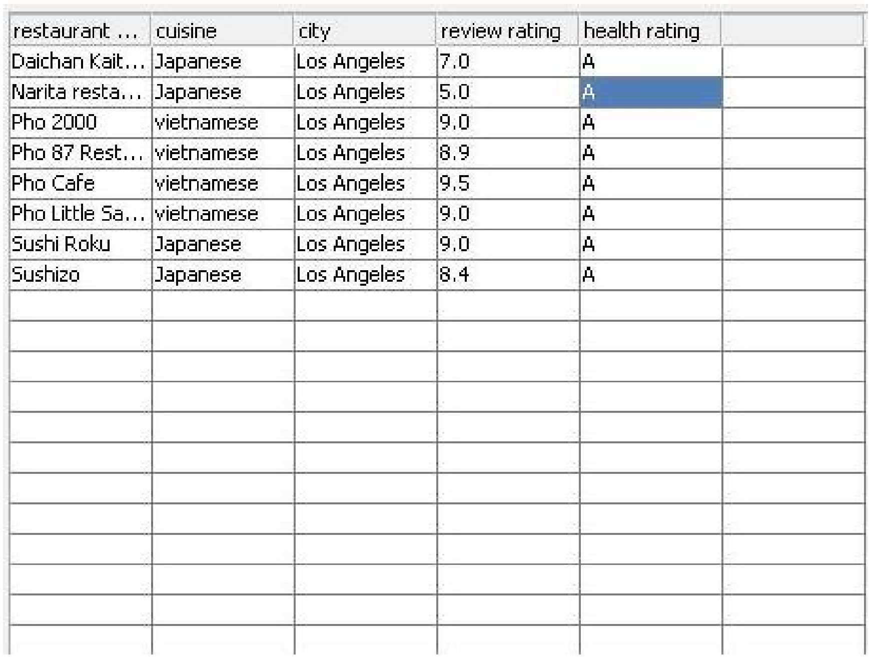
Figure 6.8: After completing the process of choosing extra attributes, the user clicks the button *Fill* and Karma will generate queries that combine and integrate the data from the new source with existing sources.



candidate: state: total_raised: total_spent: age: party:

candidate:Barack Obama
state:IL
total raised:$80,256,427
total spent:$44,169,236
age:45
party:Democrats

Figure 6.9: If the table contains geospatial information, the user can display the data on the map by choosing the tab *Table* and clicking *Map*. In this example, the state is used as a location for each data point.



restaurant ...	cuisine	city	review rating	health rating	
Daichan Kait...	Japanese	Los Angeles	7.0	A	
Narita resta...	Japanese	Los Angeles	5.0	A	
Pho 2000	vietnamese	Los Angeles	9.0	A	
Pho 87 Rest...	vietnamese	Los Angeles	8.9	A	
Pho Cafe	vietnamese	Los Angeles	9.5	A	
Pho Little Sa...	vietnamese	Los Angeles	9.0	A	
Sushi Roku	Japanese	Los Angeles	9.0	A	
Sushizo	Japanese	Los Angeles	8.4	A	

Figure 6.10: The end result of the data table generated by using Karma's data integration mode starting from an empty table.

list of possible words and search terms when given a partial keyword search. Once the user provides some sample data, Karma will translate partially-filled rows into queries that retrieve data from multiple sources, and fills in the table. Karma's novel approach of using constraints and partial plans allows the user to enter values without any knowledge about the query language or data sources. Moreover, Karma generates consistent queries that always return data. Figure 6.11 to 6.25 show the steps of the user interacting within Karma to build the table shown in Figure 6.10.

6.2.3 Augmenting data with query integration

The first walkthrough builds the Mashup by extracting data from a webpage and then augments it with existing data from databases through the help of the Karma data integration module. The second walkthrough builds the Mashup by incrementally adding

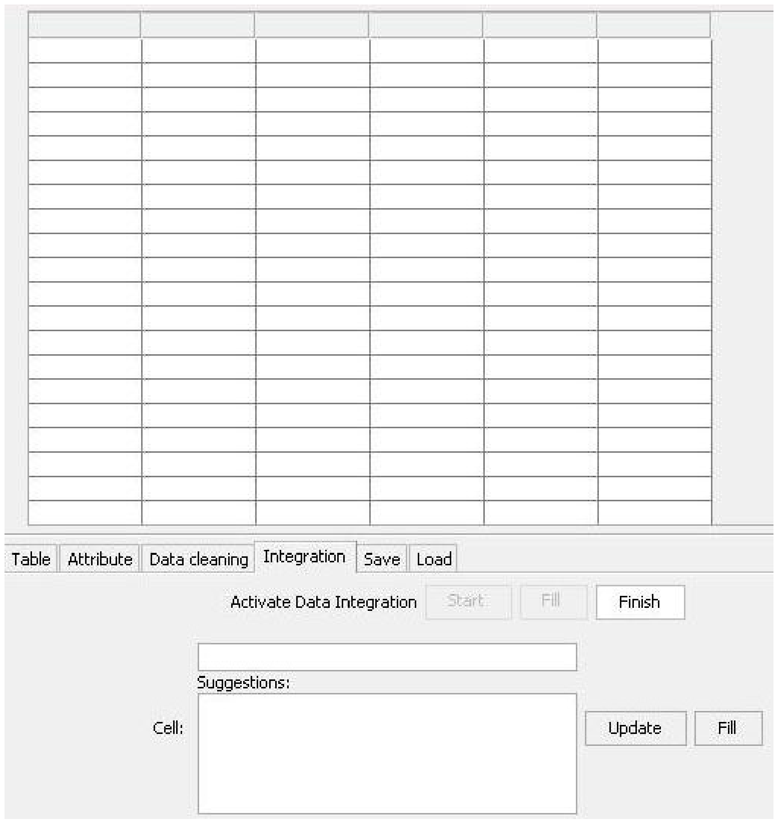


Figure 6.11: Step 1: The user begins the process of data integration by choosing the *Integration* tab and clicks *Start*.

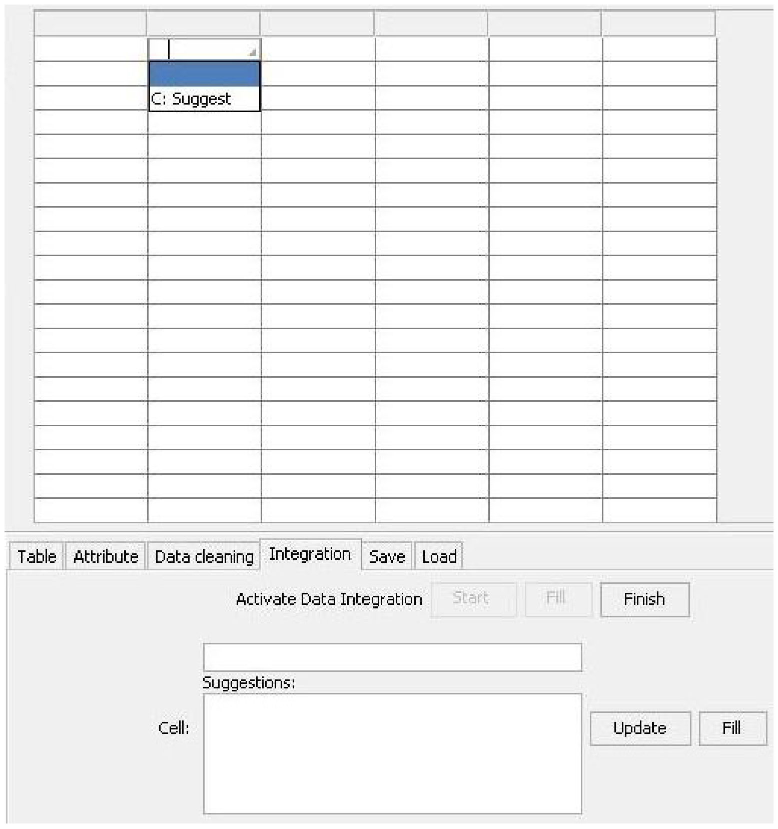


Figure 6.12: Step 2: The user can click any cell and choose *C:Suggest* (i.e., C as Command) to tell Karma that she wants a value suggestion.

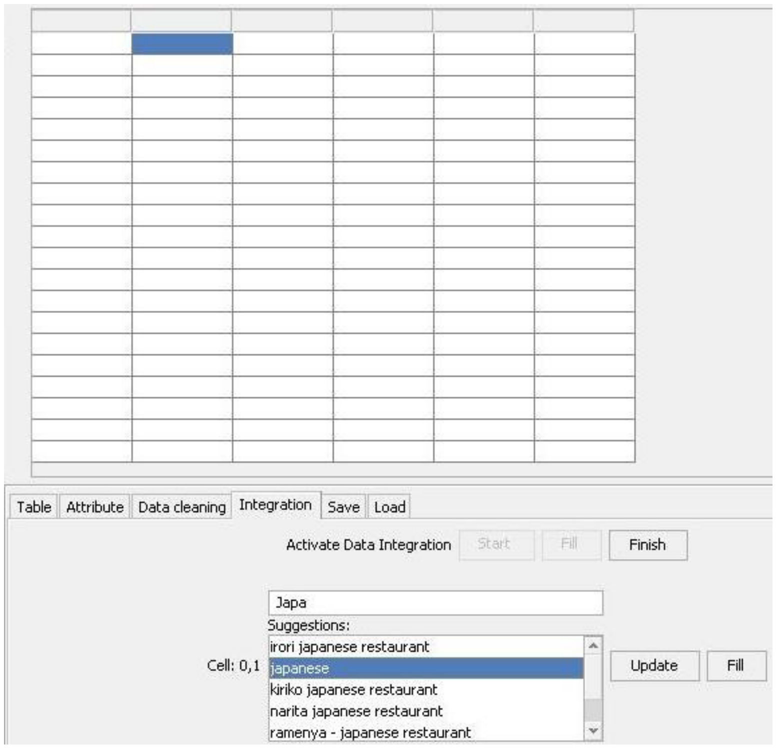
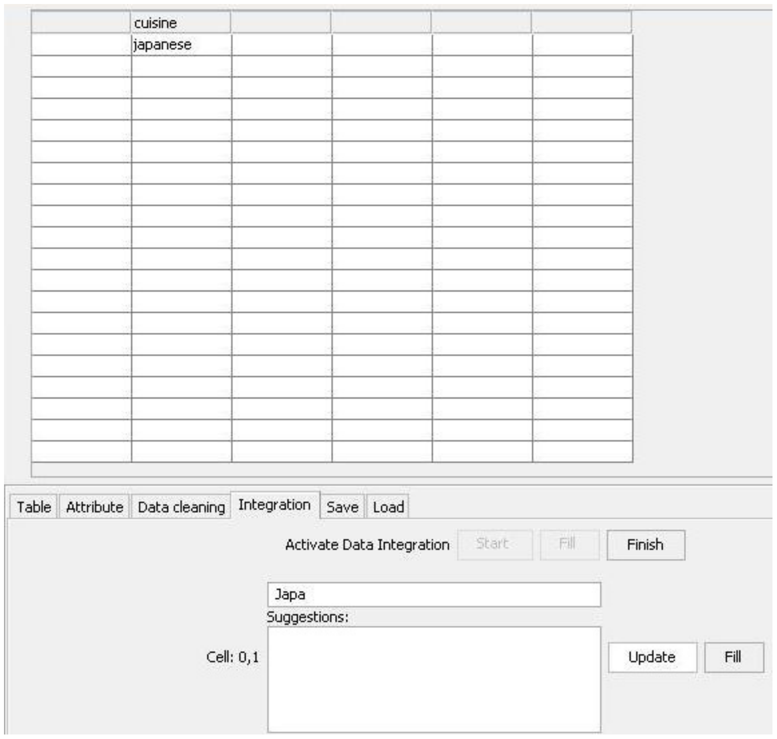


Table
Attribute
Data cleaning
Integration
Save
Load
Activate Data Integration
Start
Fill
Finish
Japa
Suggestions:
irori japanese restaurant
japanese
kiriko japanese restaurant
narita japanese restaurant
ramenya - japanese restaurant
Cell: 0,1
Update
Fill

Figure 6.13: Step 3: When the users chooses *C:Suggest*, the selected cell is recognized by Karma (i.e., *Cell: 0,1*) to the left of the suggestion box. In this case, the user decide to specify the cuisine first. As the user types, Karma eliminate choices and suggests a list of possible values to fill in a cell.



	cuisine				
	japanese				

Japa

Cell: 0,1

Figure 6.14: Step 4: Assuming that the user picks *Japanese* and clicks *Update*, the value is updated in the original selected cell.

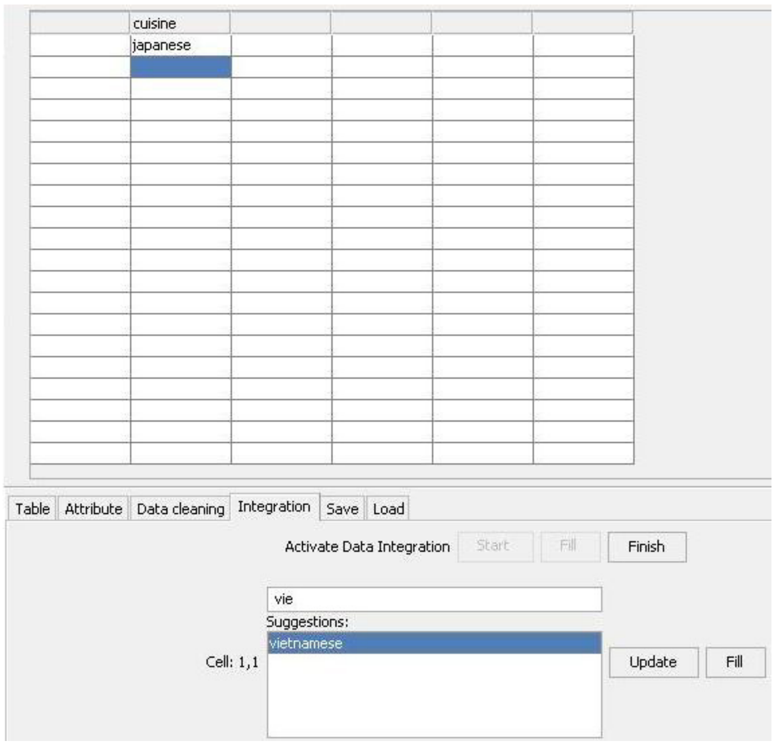


Figure 6.15: Step 5: The user now repeats the same process with the cell below cell that contains *Japanese*

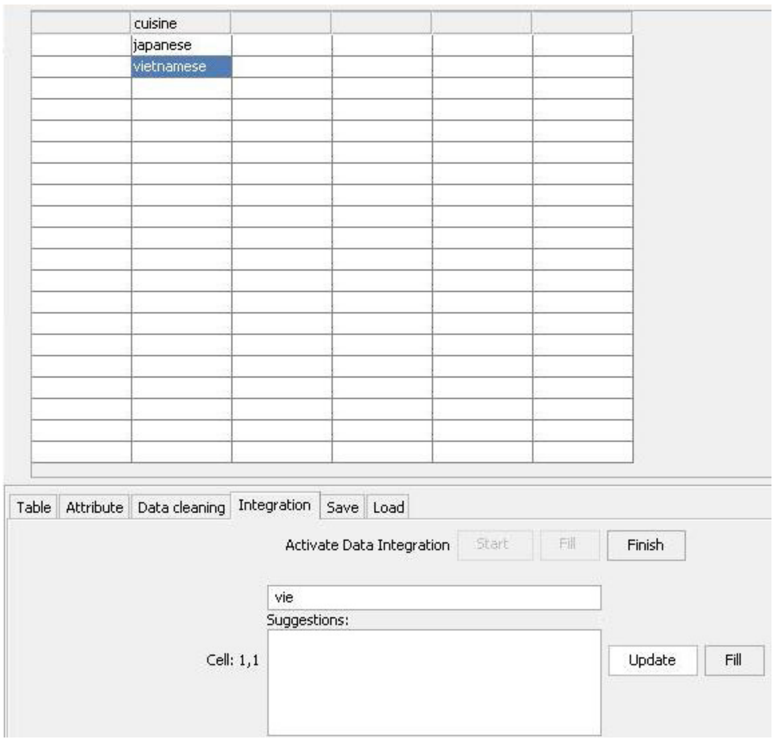


Figure 6.16: Step 6: After the user clicks *Update* the cell is filled with the value *vietnamese*. Note that at this point Karma determines that the attribute under this column can only be *Cuisine* and proceeds to automatically set the attribute to *Cuisine*.

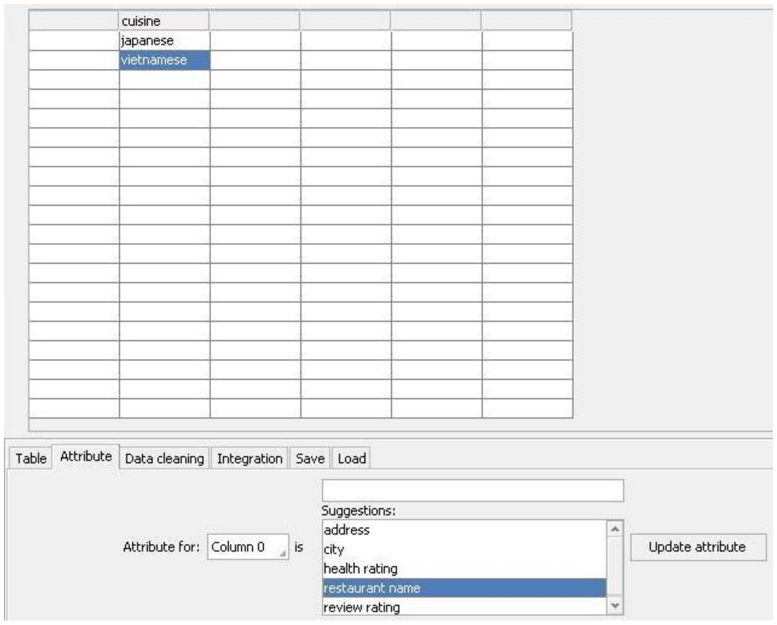


Figure 6.17: Step 7: In addition to getting a value suggestion, the user can ask Karma to suggest an attribute too. In this step, the user clicks the *Attribute* tab, picks column number, and selects from a list of available attributes. Note that only the attribute that is related to *Cuisine* will be shown. However, the user does not need to know the data sources where those attributes come from and how these atttributes are related to cuisine.

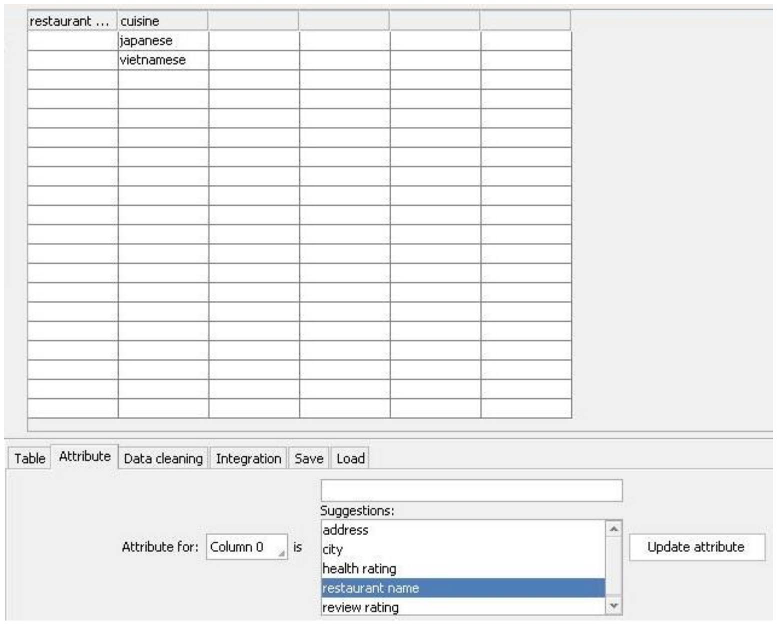
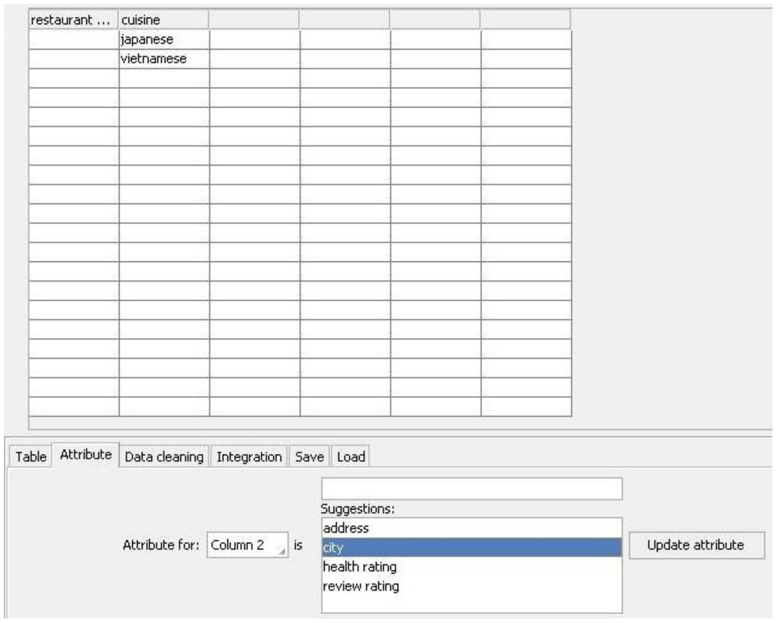


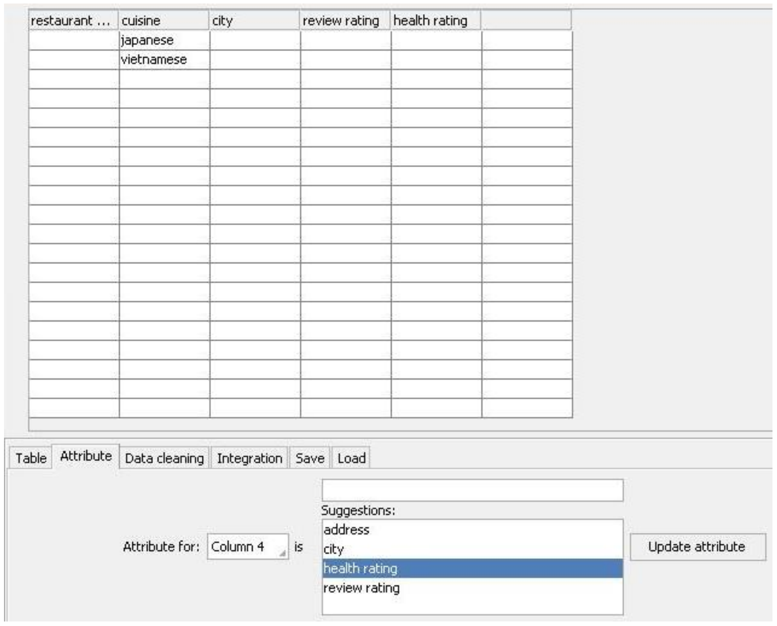
Figure 6.18: Step 8: Once the user chooses the new attribute and clicks *Update attribute*, the new attribute is set on the table.



restaurant ...	cuisine				
	japanese				
	vietnamese				

Attribute for: Column 2 is city

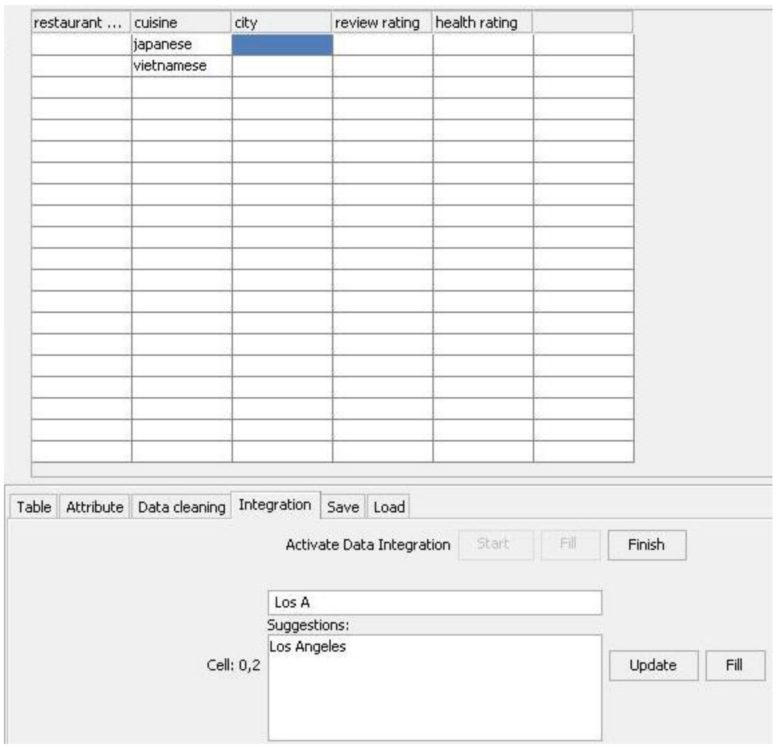
Figure 6.19: Step 9: The user can continue to select a new attribute for another column. In this case, the user wants to select the attribute *City* for Column 2.



restaurant ...	cuisine	city	review rating	health rating	
	japanese				
	vietnamese				

Attribute for: Column 4 is health rating

Figure 6.20: Step 10: The attribute selection is continued using the same process for each column until the user has five attributes: restaurant name, cuisine, city, review rating, and health rating.



restaurant ...	cuisine	city	review rating	health rating	
	japanese				
	vietnamese				

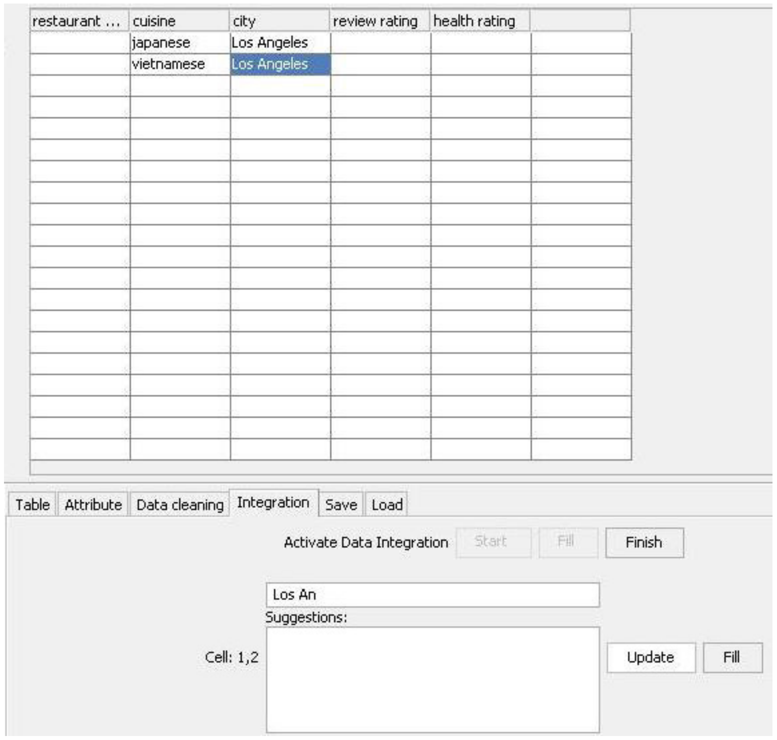
Los A

Suggestions:

Los Angeles

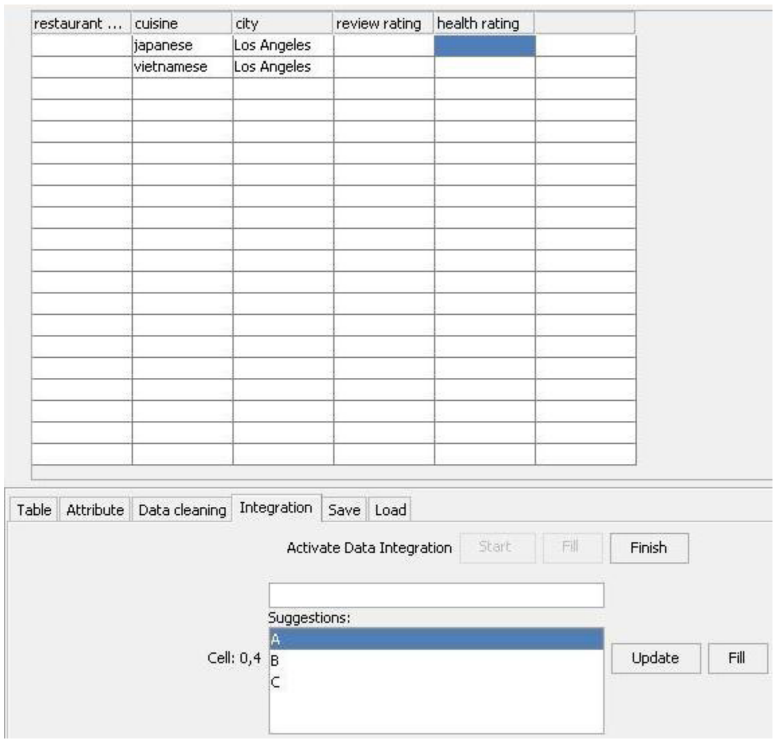
Cell: 0,2

Figure 6.21: Step 11: Since the user wants the restaurant to be in Los Angeles, the user picks a cell under the column *City* and gets a suggestion to see if *Los Angeles* is a possible value. Note that if Karma does not have a Japanese restaurant in Los Angeles, it would not suggest Los Angeles as a choice.



restaurant ...	cuisine	city	review rating	health rating	
	japanese	Los Angeles			
	vietnamese	Los Angeles			

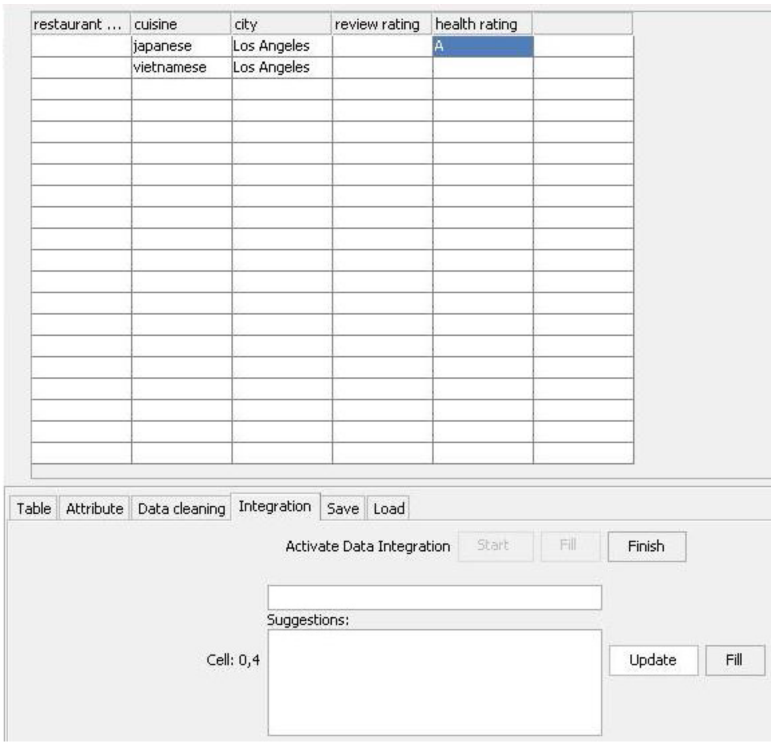
Figure 6.22: Step 12: The user fills in the two cells with the value Los Angeles as a way to indirectly specify the constraint that the restaurants in this table must be in Los Angeles, in addition to serving either Japanese or Vietnamese cuisine.



restaurant ...	cuisine	city	review rating	health rating	
	japanese	Los Angeles		🟦	
	vietnamese	Los Angeles			

Suggestions:
Cell: 0,4
A
B
C

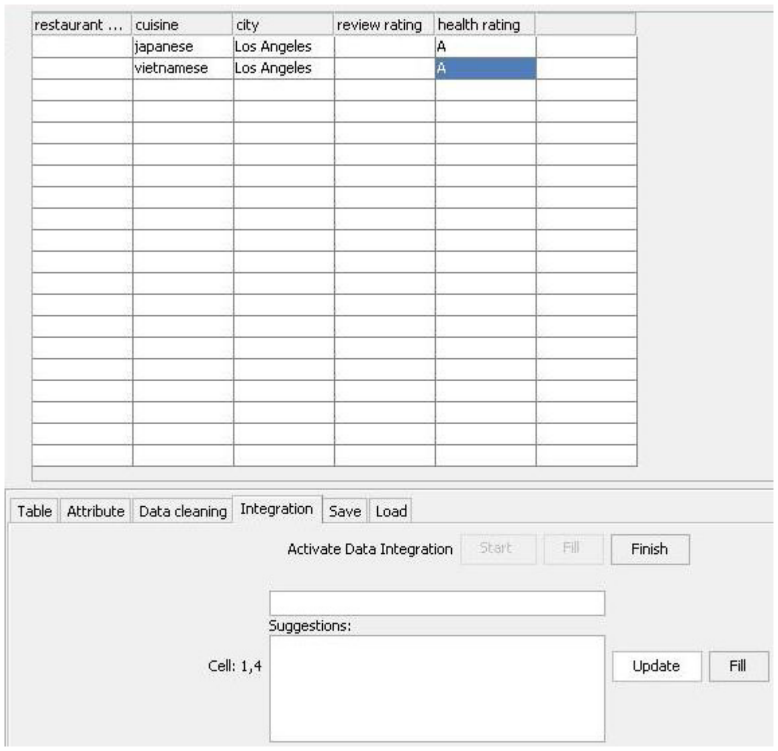
Figure 6.23: Step 13: Next the user clicks the cell under health rating to get a suggestion list. Note that since the user wants a list of restaurants with health rating equals to A, the user would select A and click *Update*. Note that Karma offers three values (A,B,C) if and only if there are Japanese restaurants for each of the health ratings.



restaurant ...	cuisine	city	review rating	health rating	
	japanese	Los Angeles		A	
	vietnamese	Los Angeles			

Figure 6.24: Step 14: After the user clicks *Update*, the cell is filled with the value *A*.

values using Karma's suggestions without extracting data from any web page. While the two scenarios start out differently, the structure of the problem during the data integration phrase is similar; both cases use Karma's databases to augment a table with more data from the databases. In the next section, we will show how the augmenting process is done starting from an empty table. However, one can imagine the case where the user starts with some data in the table (whether that data comes from new source or by using only Karma's suggestion from the database) and then continues to augment it similarly to the way it is done in the first walkthrough.



restaurant ...	cuisine	city	review rating	health rating	
	japanese	Los Angeles		A	
	vietnamese	Los Angeles		A	

Figure 6.25: Step 15: The user repeats this step to specify the health rating for the Vietnamese cuisine. Note that at this point, the user finishes selecting cells and specifying data values that express the goal constraint. At this point, the user clicks *Fill* and asks Karma to generate the query to fill the table based on the partially filled table. The result is the table shown in the beginning in Figure 6.10

6.3 Approach

In this section, we will show how to build data integration queries indirectly using examples. We will break down the approach into small cases and explain the technical detail of how Karma handles each case. First, we will explain the intuition and how indexing is used in Karma. Second, we will go into each problem case. Finally, we will discuss advantages and limitations of the query building approach.

6.3.1 Intuition

The intuition behind our approach is simple. Each data structure or process has its own constraints. By exploiting these constraints, we can narrow the search space of the solution. When the user selects a particular value, that value implies specific data sources and attribute names. These qualified data sources and attributes can be used to constrain the next value/attribute that the user may select.

Given a set of data sources in the relational form, we can *index* the data very much as search engines index web pages. We index all the values to construct a map $v \rightarrow \{(a,s)\}$ that maps the value (v) to the source (s) and the attribute (a) in that source where the value appear. A single value can be associated with multiple attribute names and data sources. For example, the mapping

Los Angeles $\rightarrow$ {(city, Zagat), (city, LA_health_rating), (hotel_name, Orbitz)}

implies that a keyword value *Los Angeles* exists in a) the data source Zagat under the attribute *city* b) the data source LA_health_rating under the attribute *city* and c) the data source Orbitz under the attribute *hotel_name*. We also index all the attribute names to construct a map of $a \rightarrow \{s\}$; for each attribute name; and we identify a list of data sources that contain such an attribute name. Once we index all the attributes and values, as the user types in a partial keyword, we can retrieve and suggest the possible set of candidates using SQL Boolean queries. Each attribute or value selected and filled by the user introduces global constraints that further limit the next attribute and value that the user can select. For example, when the user enters *Los Angeles* as one of the values, the number of data sources for that particular attribute is narrowed down to only three sources, and the number of possible distinct attributes is narrowed down to only two.

6.3.2 Indexing tables and data source definition

By indexing attributes and values into A and V hash tables, Karma can deduce information about possible values and attributes to suggest to the user.

Let:

- S: a set of all available web sources
- *att(s)*: a procedure that returns the set of attributes from the source s
- *val(a,s)*: a procedure that returns the set of values associated with the attribute a in the source s
- A: a lookup hashtable with its key and value being $a \rightarrow \{s\}$ where, 1) $\{s\} \subseteq S$ and 2) $\forall s \subset S$: a $\in$ *att(s)*
- V: a lookup hashtable with its key and value being $v \rightarrow \{$*(a,s)*$\}$ where $\forall$*(a,s)*: $v \in$ *val(a,s)* $\wedge$ S: a $\in$ *att(s)*

We will walk through the example scenario and show how we use constraints and partial plans to formulate complex queries. We will start from a simple task of building a one column table and then move to the more complicated case of building a multi-column table. We will assume that we have the following data sources in our scenario:

- **Zagat**($restaurant_name, $cuisine, $address, $city, $state, $zipcode, review_rating)
- **Asian_food_review**($restaurant_name, $cuisine, $price, $address, $city, $state,$zipcode, review_rating)
- **LA_health_rating**($restaurant_name, $address, $city, $state, $zipcode, inspection_date, health_rating)
- **EU_country_info**($country_name, language, population, gdp, date, location)

Zagat and Asian_Food_Review are both restaurant review sources. However, Asian Food Review only contains Asian food. EU_Country_Info is a data source that contains information about European Union countries. Note that EU_Country_Info may not be related to food. However, we introduce this data source to show how the values from different attributes and data sources can be overlapping, which often occurs in a repository of many sources.

Each data source has a set of attributes associated with it. For example, Zagat is a table with seven attributes. The *$* sign determines the primary key constraint, which we will explain later. Note that the source modeling step is taken care of by users when they extract and save these data sources. As a result, the attribute names defined here are uniform (e.g., we use *$address* in all the definitions rather than the possibly different attribute names that the sources may have had before alignment).

6.3.3 Single column case

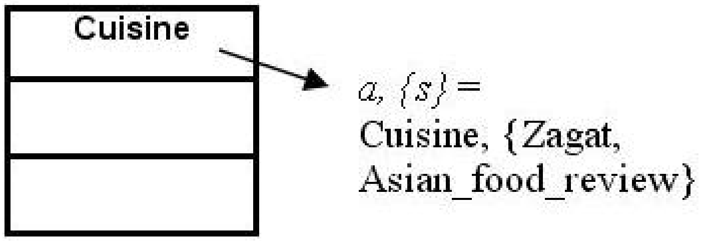


Figure 6.26: Building a single column table starting by entering an attribute.

With a blank single column table, the user can choose to enter an attribute or a value. Let us assume that the user chooses to enter an attribute first as shown in Figure 6.26. Since we do not have any constraints to start with, the attributes that can be suggested are all keys $\{a\}$ from A. Assume that the user chooses Cuisine and its associated sources are *Zagat* and *Asian_food_review*. At this point, if the user wants to enter a value, a possible list of values is constrained by the attribute selected and its associated data sources. As a result, the set of possible values that Karma can use to suggest to the user is:

$\{v\}$ = val(a,s) where $s \subset$ {s}

For example, this mathematical expression can be translated into a query as:

(SELECT Cuisine FROM Zagat)UNION(SELECT Cuisine FROM Asian_food_review)

Karma also allows the user to start by entering a value without having to specify any attribute, as shown in Figure 6.27. Since we do not have any constraint for the first value, the possible suggest candidate set is simply all $\{v\}$ from V. Let us assume that the first value the user selects is *Japanese* and its associated {*(a,s)*} is {(Cuisine, Zagat), (Language, EU_country_info)}; this means that the value *Japanese* exists in two data sources and can be associated to two attribute names - Japanese can either be a Cuisine or a Language.

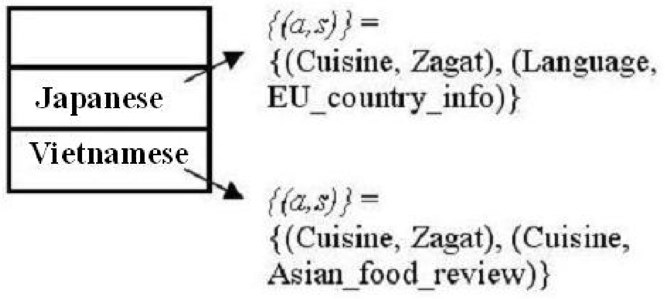

Japanese
Vietnamese
{(a,s)} =
{(Cuisine, Zagat), (Language, EU_country_info)}
{(a,s)} =
{(Cuisine, Zagat), (Cuisine, Asian_food_review)}

Figure 6.27: Building a single column table starting by entering a value first.

The value entered creates a constraint that limits the set of possible attributes and the set of values in the next step. For the sake of simplicity, we will postpone describing what the constraint is and assume that a second value *Vietnamese* satisfies the constraint.

At this point, we can describe the constraint that dictates the possible candidate set of the attribute name. If we only have *Japanese* in the table, the possible candidate set for the attribute name is {Cuisine, Language}. However, once we have both values (*Japanese* and *Vietnamese*) in the table, the possible candidate set for the attribute name must be attributes shared by both values. We formalize this idea as the set intersection constraint.

6.3.3.1 Set Intersection Constraint

We compute the attribute candidate set using the set intersection. The set intersection of the possible attribute name consistent with each row in Figure 6.27 is simply: Cuisine. At any point, if the attribute name is not defined, the possible candidate set for attribute names is:

{x} = Set intersection({a}) over all the value rows

Now, we can revisit the constraint that we ignored when the user wants to enter a second value after entering the first value *Japanese*. To enter a value, the possible value set is:

$\{v\}$ = val(a,s) where a $\in \{x\} \wedge s$ is any source where *att(s)* $\cap \{x\} \neq \{\}$

To enter a value, we first compute the set intersection (which can be more than one attribute) between all value rows. Then we retrieve a candidate set from any data source that contains any attribute in the set intersection. The query to retrieve the candidate set for the second value (*Vietnamese*) is:

(SELECT Cuisine FROM Zagat)UNION(SELECT Language FROM EU_country_info)

After the user chooses *Vietnamese* as a second value, if the user wants the enter the third value, the query to retrieve the candidate set will be:

(SELECT Cuisine FROM Zagat)UNION(SELECT Cuisine FROM Asian_food_review)

Note that

- The query for the third value does not include the attribute Language from the source EU_country_info because the attribute Language does not belong to the updated set intersection.

- As the user enters the attribute or values, the possible choices are becoming more limited.

- Once the choices are narrowed down to only one value (i.e., the set intersection of the attribute between *Japanese* and *Vietnamese* = Cuisine), Karma will fill that attribute/value in the table automatically. In our case, once the user selects *Vietnamese*, Karma will also update the unfilled attribute name to be *Cuisine*.

6.3.4 Multiple column case

The multiple-column case is more complex, because a value entered in one column can affect how Karma suggests attributes and values in other columns. First, we will introduce the concept of *reachable* attributes. Then we will describe partial plans. We will show different examples to explain each concept.

6.3.4.1 Computing reachable attributes

In traditional databases, we can link different tables together using the join operation. Depending on the join condition, it is possible to create a successive chain of tables. Figure 6.28 shows an example of how tables in a database can be linked. Given an employee ID, we could retrieve the following attributes using join conditions through foreign keys: SSN, name, address, phone number, latitude, and longitude

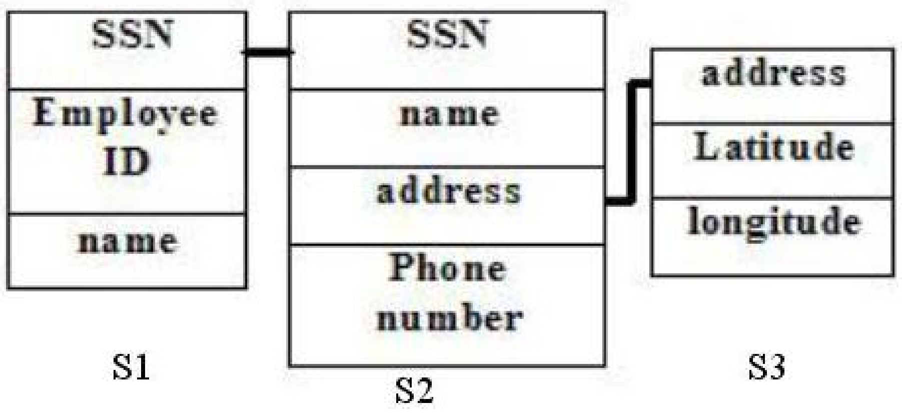


Figure 6.28: Joining condition through foreign keys in traditional database

We define a *reachable* attribute as an attribute that can be reached from a particular data source (e.g., longitude is reachable from S1). If we have a well-defined database like the one shown in Figure 6.28, join conditions between tables can be composed over foreign keys. Joining two tables using non-foreign key attributes (e.g., name) is possible, but the result generated may not make sense. For example, there might be a record with name *John Smith* in S1 and S2 who are completely different people with different SSNs.

In Karma, we use a primary key constraint. The set of attributes with the $ in the data source model acts as primary keys. For example, the primary key constraint for the *LA_health_rating* source is: $restaurant name, $address, $city, $state, and $zipcode.

What the constraint means is that if we want to retrieve inspection date and health rating, the join condition must be over restaurant name, address, city, state, and zipcode.

In many web sites, to retrieve information, we need to fill out a web form. For example, to get a health rating for a particular restaurant, we need to fill out a form (e.g., restaurant name, address). For a wrapper to retrieve the data from these sites, it will also need this information, hence the input requirement. In Karma, we retain this input requirement information and use it as the primary key constraint. The reachable attribute definition allows us to handle the case where the user wants to enter a new attribute in a multi-column table.

Continuing from the single column case (Figure 6.27), if the user wants to enter the new attribute as shown in Figure 6.29, the possible set of valid attributes must be reachable from each row in the table (i.e., both the *Japanese* row and the *Vietnamese* row). For example, the attribute price is only reachable from the *Vietnamese* row, because *price* is the attribute that belongs to the Asian_food_review data source. If we allow price

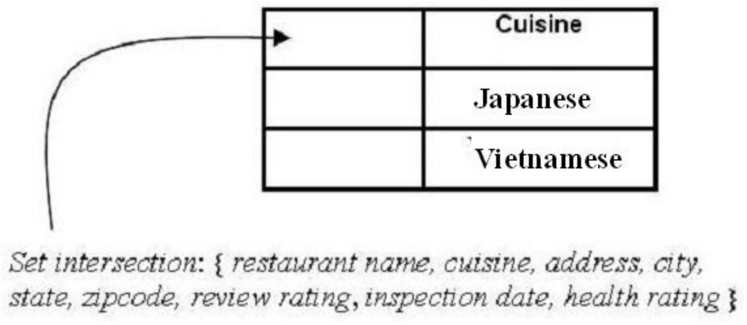

Cuisine
Japanese
Vietnamese
Set intersection: { restaurant name, cuisine, address, city, state, zipcode, review rating, inspection date, health rating }

Figure 6.29: Entering an attribute in the new column requires a computation of the set intersection of "reachable" attributes between each row.

to be used as the new attribute, we will not be able to suggest any value for the *Japanese* row, because price is not reachable from that row (since the first row does not contain Asian_food_review).

As a result, the first constraint for entering a new attribute in the table is: the set intersection of *reachable* attributes of all partially filled rows

However, simply matching the primary key constraint between data sources does not guarantee that all attributes in the set intersection will create a join condition that does not return the empty value. The second constraint that Karma imposes is: each attribute in the set intersection of the *reachable* attributes set must produce a non-empty suggested value set.

This can be accomplished by executing multiple queries through partial plans as if Karma wants to suggest possible values for each attribute from (1) in each partially-filled row, and eliminate the ones that produce an empty set from *reachable* attribute list. This constraint allows Karma to guarantee that for each empty cell in a non-empty row, Karma can fill those cells. Note that it is also possible to configure Karma to relax these two constraints to allow a partially filled table with some missing values in case where we do not have enough coverage from data sources to be able to suggest at least one value in each row.

6.3.4.2 Partial Plans

Assuming that the user selects *restaurant name* for the new attribute in Figure 6.29, the user might want to (a) have Karma fill the blank cells in the table. (i.e., find all the restaurants with Cuisine = Japanese or Cuisine = Vietnamese) or (b) fill in some cells under the column *restaurant name* by letting Karma suggest the available choices (as in Figure 1). In both instances, Karma uses partial plans to help evaluate what data should be used to either fill the table or suggest values to the user.

Each individual row in the Karma table contains a partial plan in the form of a tree that keeps track of selected values and attributes. Each row can contain more than one partial plan depending on the number of data sources. For example, the second row in Figure 6.27 would contain two partial plans (one plan using Zagat, and the other using Asian_food_review).

The partial plan tree can easily be translated into queries to retrieve the candidate values to suggest to users. First, we will show the final tree (Figure 6.30) that Karma constructs for the first row from the table in Figure 6.25. Then, we will explain: (1) node types and parent types (2) how we can use the tree to evaluate the possible suggestion set that can be used to suggest a value for a cell or fill the whole table, and (3) how we construct such a tree.

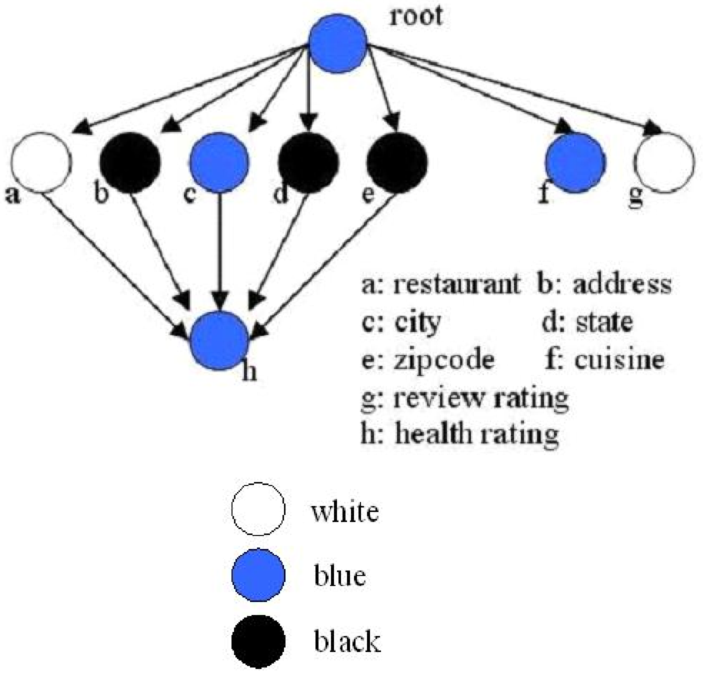


Figure 6.30: The partial plan for the first row of the table in Figure 6.25 that includes joining between two data sources according to the primary key constraint.

6.3.4.3 Node types and parent types

Each node represents a cell in a row, except for a root node, which is used as a starting point. There are three types of nodes: a value node (blue), a place holder node (white), a hidden node (black). The value node is the node where the user has already specified a value in that cell (i.e., *Japanese*, *Vietnamese*). The place holder node is a node where the user has not selected any value yet, but the attribute for that column is already specified. For example, an empty cell on the first row under the attribute *restaurant name* in the table in Figure 6.25 corresponds to a place holder node. The hidden node is a node that is required as a part of the joining condition through the primary key constraint, but is not selected into the table by the user (nodes b,d,e). Each node will contain a triplet of attribute, source, and value. For example, node f will contain f(a,s,v) = (cuisine, Zagat, Japanese), while node **'a'** (a place holder node) will contain a(a,s,v) = (restaurant name, Zagat, _PLACE_HOLDER). When the user first enters a value, we designate its corresponding data source as a starting source. Any node with its **'s'** equal to the starting source has the root as its parent, while a node that is the result of a joining data source will have multiple parents. Those parent nodes are essentially nodes with primary key attributes.

6.3.4.4 Tree evaluation

The partial plan tree can be used to suggest a particular set of available values by translating it into a query for a particular node. This process can also be used to verify wheter a particular attribute will produce a non-empty suggest value set or not. To translate a tree into a query, we use the following rules:

1. A value node implies a value equal '=' condition.

2. A place holder node can only be included in the SELECT part of the query

3. A node with multiple parents implies a join condition over its parents.

For example, a possible candidate set for the attribute *restaurant name* (a place holder node *a* in Figure 6.30) in the first row is:

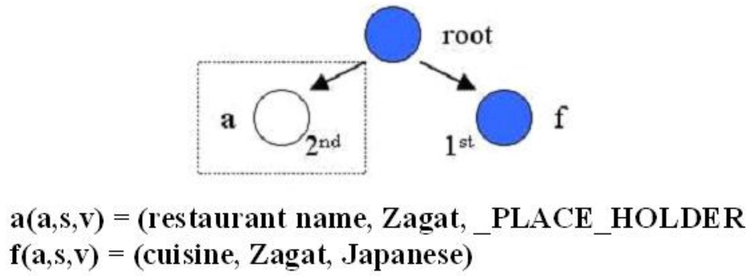
```
SELECT DISTINCT Zagat.‘restaurant name’
FROM Zagat, LA_health_rating as L
WHERE Zagat.city = “Los Angeles” AND
Zagat.cuisine = “Japanese” AND
L.’health rating’ = “A” AND
Zagat.’restaurant name’=L.’restaurant name’ AND
Zagat.’address’ = L.’address’ AND
Zagat.’city’ = L.’city’ AND
Zagat.’state’ = L.’state’ AND
Zagat.’zipcode’ = L.’zipcode’;
```

The first three conditions are the constraints imposed by value nodes (rule 1), while the rest are the constraints imposed by joining conditions (rule 3).

Note that we can also create a query that fills the whole table by retrieving the possible set by including: city, cuisine, review rating, and health rating in the SELECT part of the query above.

6.3.4.5 Tree construction

In our scenario, the user starts in the single column table with an empty table and selects *Japanese* in a cell. That is when we add the first value node f into the tree. On the other hand, if the user starts with selecting an attribute first, we will add a place holder node instead, because the value for that cell has not been specified by the user yet.



a(a,s,v) = (restaurant name, Zagat, _PLACE_HOLDER
f(a,s,v) = (cuisine, Zagat, Japanese)

Figure 6.31: The first row partial plan generated by Karma as the user enters attributes and values.

To add the first node, the set intersection constraint must be satisfied. Note that it is possible that (a) the size of the set intersection is more than one (the value selected corresponds to more than one attribute name), or (b) the corresponding data sources are more than one. In such cases, we permute over each data source to create multiple partial plan trees for each row, where each plan uses one data source. As the user enters more values that result in the elimination of attributes/data sources according to the set intersection constraints, Karma removes the partial plan trees that refer to the eliminated attributes/data sources. For example, (Language, EU_country_info) is eliminated when we compute the set intersection constraint in Figure 6.27. From the second node on, we are dealing with a multiple-column table case. For example, to add a second node (node **'a'**) in Figure 6.31:

- Compute the set intersection of the reachable attributes of all the rows. Then, keep only reachable attributes that return non-empty suggest value sets.

- At this point, we have a set of attributes that Karma can suggest to the user. If the user selects the attribute (as in Figure 6.29), create a place holder node (node **'a'** in Figure 6.31). If the attribute is in the same data source as the starting source, add the new node as the child of the root. However, if the attribute is in a different source, create necessary hidden nodes according to the primary key constraints and set the new node as the child of those hidden nodes.

- On the other hand, if the user decides to enter a value instead of selecting the attribute in Figure 6.29, permute the partial plan for the set intersection attribute set in (1). At this point, we will have multiple partial plans similar to that of a plan in Figure 6.31, although the attribute of the node **'a'** will be permuted over all set of reachable attributes. We can evaluate each plan to retrieve the possible values from node **'a'** of all partial plans. The union of the result will be used as a suggest set for the user. Once the attribute can be determined, remove the partial plans that do not correspond to that attribute.

We have demonstrated how the set intersection and the reachable constraints can be applied to partial plans. This approach allows the user to select any cell in the Karma table, get a list of suggestions, and, once satisfied, tell the system to fill the remaining part of the table with the data that satisfies all the constraints.

6.4 Discussion

Our approach is based on the idea that every problem has a structure that dictates the constraints. Once we find the constraints, we use them to limit the search space of the solution. Our work illustrates how past approaches [66] underutilize the information from the structure of the problem that, once exploited, can reduce the user's time and knowledge requirement to perform a task. We describe the advantages and limitations or our approach below.

6.4.1 Advantages

The advantages of our approach are the following:

- *Consistent Query Generation*: The problem in many query systems is that the user can formulate a query, but that query is not guaranteed to return a non-empty result set. Our approach uses the set intersection and reachable constraints to ensure that if we suggest a new attribute, we can fill every row in that column with non-empty values. Since we only allow the user to select attributes/values from the list of possible attributes/values, there is no way a user can make a mistake.
- *True Query by Example*: Our approach is a true query by example. By abstracting away the data sources and their linking information, the user only needs to enter values to create a complex query.

6.4.2 Limitations

The limitations of our approach are the following:

- *Primary Keys Requirement*: As discussed in the multiple-column cases, we need primary keys between data sources to compute reachable attributes. For web sources, we can use the wrapper input requirement as the primary key constraint. However, if we want to integrate a data source that is not a wrapper, we will need an expert to label the primary key of that particular source.

- *Scaling*: In terms of indexing all the attributes and values into A and V lookup tables, we believe that it is feasible on the WWW scale; Google Suggest is one such example. Our limitation in terms of scaling belongs to computing reachable attributes. Given enough attributes and the right set of primary key constraints, we can link data sources in a very long chain. Since our suggest set relies on evaluating and verifying partial plans to eliminate the ones that return empty data, the size and the number of partial plans can be exponentially high. Currently we limit the length of the chain to be only two (the partial plan tree can only have a maximum depth of three).

Chapter 7

Related Work

This chapter surveys previous work related to Mashups. First, I discuss existing Mashup building tools from both academia and industry. Then I discuss each research area based on how I segment the Mashup building problems: data extraction, source modeling, data cleaning, and data integration.

7.1 Mashup building tools

There exists an array of Mashups building tools. However, one predominant approach adopted by many tools is the widget approach. As a result, I have dedicated a subsection to it. I other approaches in separate subsection. Finally, I show how Mashup tools that aim to support average users address each Mashup building issues.

7.1.1 Widget Approach

The Mashup tools in this category use a widget paradigm as their basis. This approach probably originates from a merger between visual programming lauguage [10] and dataflow programming [55]. A visual programming language allows users to specify programming elements graphically instead of textually [1]. A survey of existing visual language research can be found at [10]. Dataflow programming views programming as a graph of interconnected operations [1]. In the widget paradigm, a user selects a widget, drops a widget onto a canvas, customizes the widget, and specifies how to connect that widget to other widgets, creating a connected graph.

The following systems use the widget paradigm: Yahoo's Pipes, Microsoft's Popfly, Marmite[62], JackBe (http://www.jackbe.com), Bungee Labs (http://www.bungeelabs.com), Proto Software (http://www.protosw.com), and IBM QED Wiki (http://www.ibm.com). Yahoo's Pipes, Microsoft's Popfly, and CMU's Marmite[62] are similar structurally in terms of their approach. They rely on the widget paradigm where users select a widget, drop a widget onto a canvas, customize the widget, and specify how to connect widgets. The difference between each system is the number of widgets (e.g., 43 for Pipes and around 300 for Popfly), the type of widgets supported, and the ease of use. For example, Marmite and Popfly will suggest possible widgets that can be connected to existing ones on the canvas, while Pipes will rely on users to select the right widgets. Compared to these systems, Karma uses a unified paradigm that does not require users to locate widgets or understand how each widget works.

Bungee Labs (www.bungeelabs.com), IBM's QED wiki (www.ibm.com), and Proto Software (www.protosw.com) are example Mashup tools for enterprise applications. These tools also use widgets to support most Mashup building functionality. As a result, experts are required to use them because configuring these widgets to handle industry strength applications is difficult.

7.1.2 Paradigm Comparison

While it would be interesting to report detailed comparison between each system that uses the widget approach and Karma, it is impossible to obtain and test each software since a) some are commerical applications that require purchasing and b) some are research prototypes that cannot be obtained. However, we can compare Karma with these systems based on their approach paradigm. Karma differs from the widget paradigm in three different aspects: interface, interaction, and problem segmentation. The differences between the Karma paradigm and the widget paradigm are listed below:

- Interface: In the widget paradigm, a user builds a Mashup by composing a connected graph of widgets on a canvas. Except for Marmite, users of these systems have to re-execute all the widgets while debugging a Mashup to see any intermediate result. Also, as the Mashup becomes more complicated, the user has to navigate through a complex graph to debug or add new widgets. In Karma, the user builds a Mashup by trying to populate one table with data. As a result, the user sees the current state of a Mashup from the table and can shape the final Mashup by filling the data in the table.

- Interaction: In the widget paradigm, the user needs to customize a widget and specifies how to connect widgets. This can be challenging, and it requires the user to understand how each widget works. In Karma, the user specifies an example in the form of data and has Karma indirectly infer operations from the sample data.

- Problem segmentation: In the widget paradigm, there is no clear standard for separating each Mashup building problem type. Each system groups widget selection window based on what the designer sees fit, which can be confusing and unintuitive for users. For example, Yahoo's Pipes has ten types of widgets: Sources, User inputs, Operators, Url, String, Date, Location, Number, My Pipes, and Deprecated. In Karma, the user can trigger different modes (i.e., data extraction, source modeling, data cleaning, and data integration) based on the Mashup building stage she is in.

7.1.3 Other approaches

Besides the widget approach, there are other approaches to building Mashups. These approaches are covered below:

Simile [30], the earliest system developed at MIT, focuses mainly on retrieving the data from web pages using a DOM tree. Similar works as a Firefox plugin where users can click on a particular text on a website and save it in the repository. If that data element belongs to a list, the rest of elements are also highlighted and extrated. After extracting data from web pages, users can also tag sources with keywords that can be searched and published later.

Dapper improves over Simile by providing an end-to-end system to build a Mashup. In Dapper, users are led step by step linearly through an interaction screen, where they address data extraction, source modeling, and data integration problem. However, they still have to do most of the work manually. Dapper also provides only one cleaning operation that enables users to extract a segment of text (similar to Javas substring). Compared to Simile and Dapper, Karma extends the DOM tree approach to support more data structures and extraction from detail pages.

Potluck [31] is a recent Mashup tool developed at MIT. Potluck assumes that web sources already have RDF descriptions that enable easy extraction and provide infrastructure that addresses souce modeling, data cleaning, and data integration. One could use Simile (e.g., for data extraction) in conjunction with Potluck to build Mashups. Potluck does provide suggestions to users during data cleaning. Compared to Karma, however, Potluck lacks many automated features that enable users to address Mashup building problems quickly. For example, Potluck's users need to manually map attributes between sources and specify data integration without any system suggestion.

Intel's MashMaker[21] took a different approach where its platform supports multiple levels of users. In MashMaker, expert users would do all the work in each area. A normal user would use the system by browsing a page (e.g., Craigslists apartment), and MashMaker will suggest data from other sources that can be retrieved and combined (e.g., movie theaters nearby) with data on the user's current page. Note that MashMaker supports only web pages that are already extracted through Dapper. Compared to Karma, MashMaker limits choices for normal users to pages that exist in Dapper and data integration plans that have already been specified by experts.

Cards [20] views a Mashup as a collection of *cards*; a different way to define a tuple of data. Its users extract data from a website and store it in a card. Users can model relationships between sources and indirectly specify a way to integrate them by drawing a link between attributes, between cards, or between a card and a web site. A collection of cards can also be combined to create Mashups. While the concept of cards and linking cards together is new, Cards do not cover data cleaning. In Karma, data cleaning is supported. Moreover, Karma can suggest sources to link to a newly extracted data source automatically from previously built Mashups.

D.Mix[27] and OpenKapow (openkapow.com) allow users to cut and paste data from web pages to be used later. However, both systems assume some level of expertise in programming in HTML and Javascript. On the other hand, Karma does not require users to understand any programming language.

Google MyMaps allows users to create and import map points from limited sources. However, the process of adding a map point is often done manually. Aside from Google MyMaps, Google also has its own Mashup Editor (editor.googleMashups.com). However, it is aimed at programmers.

Agent Wizard[58] lets users create a Mashup by answering a series of questions. Its users build Mashup incrementally in a bottom up manner. As users answer more questions, more operations are added to an overall plan that extracts, filters, and integrates data. However, agent wizard has two disadvantages compared to Karma. First, as a Mashup gets more complicated, Agent Wizard's users need to answer many more questions. Second, they also need to understand how to specify filter and join conditions. In Karma, those steps are done indirectly through data samples given by users.

7.1.4 Mashup Buildling Problem Coverage

Table 7.1 shows a detailed comparison of the Mashup tools. I have only selected tools that are targeted to nonexpert users for comparison.

Table 7.1: Approach comparison between different Mashup tools segmented by problem areas.

System	Data Retrieval	Source Modeling	Data Cleaning	Data Integration	Mashup Type Supported
MIT's Simile	DOM	Manual	N/A	N/A	1
MIT's Pot Luck	RDF	Manual	PBD	Manual	1,3,4
Dapper	DOM	Manual	Manual	Join only	1,2,4
Yahoo's Pipes	Widgets	Manual	Widgets	Union only	1,2,3
MS's Popfly	Widgets	Manual	Widgets	Widgets	1,2,4
CMU's Marmite	Widgets	Manual	Widgets	Widgets	1,2,4
Intel's Mashmaker	Dapper	Manual	Widgets	Expert	1,2,3,4
Google MyMap	Widgets	Manual	N/A	Union only	1,2
Agent Wizard	Q/A	Q/A	Q/A	Q/A	1,3,4
Cards	DOM	Manual	N/A	Manual	1,2,4
Karma	DOM	Database	PBD	PBD	1,2,3,4

The terminology of how each tool handles each problem area is shown below:

- **Database** means that databases are leveraged to help generate suggestions to assist users in a specific problem area.

- **DOM** means that the document object model approach is used to handle extraction. However, there is also a varying degree of how each system utilize the DOM. For example, Simile might only uses the DOM as is. However, Karma might build on DOM to enchance data extraction on some difficult cases.

- **Experts** means that an expert is required to solve that specific problem area.

- **N/A** means that the specific problem area is not addressed or supported or the information about it cannot be found.

- **Join only** means that only database join is supported in the data integration step.

- **Manual** means that the specific problem area is supported, but a user needs to do it manually. For example, a manual approach in source modeling means that the user has to specify the relationship between data sources.

- **PBD** means that the technique Programming by Demonstration is used to solve that specific problem area.

- **Q/A** means that the user has to answer one or more questions to solve a specific problem area. The answer might also involve specifying a join condition between sources.

- **RDF** means that the data extraction step assumes that a web source has an RDF representation which allows easy retrieval of data.

- **Widgets** means that a widget must be selected and customized to tackle that specific problem area.

- **Union only** means that only database union is supported in the data integration step.

Recall from Chapter 2 the four types of Mashups that Karma supports:

1. **Type 1: One simple source:** Extracting a list of data from a single data source

2. **Type 2: Combining data points from two or more separate sources:** Database union of two sources

3. **Type 3: One source with form:** Extracting a list of data from a single data source that has HTML form input.

4. **Type 4: Combining two or more sources using a database join:** The data from the one source needs to be joined with data from another source.

   As shown in Table 7.1, the only Mashup building tools that support all four types of Mashups is Intel Mashmaker. However, Mashmaker requires an expert user to customize at least Mashups of type 3 and type 4. Most systems support up to three types of Mashups, where type 3 seems to be ignored because of the complexity of capturing HTML forms.

7.2 Data Retrieval

Research in data extraction aims at converting web pages into a relational data table that can be integrated into databases or used by agents. Research work in this area can be divided into two categories. The first category is automatic extraction where an agent tries to extract the data from a web source automatically. The second category requires some form of user input. This category can be furthered divided into a) a machine learning approach, and b) a DOM approach.

Earlier research work attempts to extract web sources that have structured data such as lists or tables. For example, RoadRunner[16] and Adel[37] can extract those lists and tables automatically. However, those systems cannot extract data from semi-structure sources. Stalker[46] uses active learning where users label training examples for the system to learn rules to extract data from semi-structure sources. Stalker's approach is semi-supervised and requires users to label sample data. When the structure of a web source is complicated, the amount of labeling required increase propotionally.

DOM is a tree-like structure that a web browser uses to interpret a HTML page. This structure is readily available and accessible from a web browser. I have provided a detailed discussion about the DOM in chapter 3. Simile[30], Dapper, PLOW[4], and Cards[20] use a document object model (DOM) tree to help identify the list and table structure of the data to extract.

Karma uses the DOM tree approach like[30] to help identify extracted data and lists. However, Karma improved on the DOM to handle difficult extraction cases. As discussed in Chapter 3, Karma handles a) extracting a list within a list, and b) extracting and linking data across multiple detailed pages.

7.3 Source Modeling

The problem of schema matching aims to find the mapping of elements between two sets of schemas[51]. The problem of source modeling aims to understand what a new source does in terms of existing sources. These two problems are closely related, because understanding what a new source does might require mapping attributes between the new source and existing sources. In addition, both problems employ similar techniques, such as bootstrapping overlapping data and string similarity.

The early research[50] defines the problem of source modeling as the category translation problem [11]. However, this work made a simplified assumption about the property of data sources (i.e., no binding constraints). Recently, the work by [29, 18, 11] improves the process of source modeling using various techniques to find semantic similarities between sources.

In the schema matching domain, [51, 26] provide good surveys of the work done in the area. Initially, researchers centered their efforts on 1:1 matching (i.e., matching one attribute to another attribute). Early works, such as TranScm[45] and Artemis[7], use a rule-based approach to determine how to map attributes together. Later on, Semint[38] and ILA[50] employed machine learning techniques to learn matching attributes from training samples. LSD[19] provides the framework to support multiple learners to achieve better matching accuracy. Currently, the n:m matching problem, where one or more attribute can be mapped to multiple attributes, is addressed in [64, 18].

As stated earlier, the work in this area aims to generate the candidate matches automatically. However, the accuracy of existing approaches (50-86%)[18] is still not efficient enough to solve the problem automatically. My thesis does not aim to invent a new technique to solve the problems of source modeling and schema matching. Karma uses simple techniques, such as string similarities comparison, and rely on users to interactively narrow the candidate match. Currently, Karma only supports 1:1 mapping, but it can be extended to support complicated matchings under the same interaction paradigm.

7.4 Data Cleaning

A review of available commercial data cleaning tools is covered in [12]. Most commercial tools focus on the process of Extraction-Transformation-Loading (ETL) through a scripting language; only trained experts can use these tools.

Potter's Wheel[52] takes an interactive approach to data cleaning by letting users specify the clean data and then inducing transformation language adaptation techniques described in [2, 14, 33]. Karma also lets users specify the clean data for transformation induction. Additionally, Karma suggests cleaning values based on the overlapping of new data and existing data in the databases.

The record linkage problem is sometimes referred as data cleaning[40]. Specifically, record linkage aims at determining if two records from different sources relate to the same entity. A good survey of the work in this area is covered in [61]. My approach does not aim to match two records. Instead, once the matching attribute is identified, only values under those two attributes will be matched. However, record linkage techniques, such as string similarities [9], are leveraged in Karma's interactive process.

Data extraction from unstructured information [44] can also be considered data cleaning since the process of the extraction is the conversion from unstructured text into a structured database. Often a reference set is used in the process of extraction and cleaning in this type of work [3, 43]. In Karma, every aligned source is considered as a potential reference set for cleaning, while existing work only has a specific set of reference sets used for cleaning.

In comparison, my approach leverages existing techniques that help users clean data interactively through the source modeling process. The process of indirectly gaining training samples to induce transformations is a huge improvement over Potter's Wheel, where users need to provide training samples manually.

7.5 Data Integration

The data integration module in Karma exploits the structure of a table to enable constraint formulations that create database queries. The approach of exploiting structure to solve a problem has been used in many research areas, such as operations research [48] and constraint satisfaction problem [41].

The goal of the data integration research is to allow casual users to access, locate, and integrate data using a uniform query interface [26]. The general survey of the data integration field in the past twenty years can be found at [25, 26].

When the data integration field was still in early stages, Ullman proposed the Universal Relation concept[59] that aims to reduce the complexity of writing queries by eliminating source specification from the query. By abstracting the whole database into one single source, users can write the query without having to specify data sources and join conditions. However, Karma abstracts both queries and sources from users, so they never have to write any query.

Query by example (QBE)[66] is a two dimensional query interface. Users formulate the query by working in a table similar to Karma. Filtering and join condition can be specified through a more simplified query language. However, the users still need to select sources manually, and the query created is not guaranteed to return results. On the other hand, Karma induces the sources to use automatically and guides users to fill in only valid values. As a result, users do not need to know about data sources and queries generated by Karma will always return results.

RABBIT[60] and HELGON[23] use the technique of retrieval by formulation. This technique allows users to form partial descriptions of a goal. Based on the partial description, these systems retrieve data that closely matches the description. The users can criticize the data and their criticism will be used to help reformulate new queries to retrieve new results. However, the users need to know which source schema concepts to use in order to form the partial description. In contrast, Karma's users can guess concepts and values to form the partial description.

The work in graphical visual query language [6, 28, 49] uses a graph to represent a query. Users can construct graphs of a query by specifying nodes (e.g., sources and attributes) and links (e.g., filtering conditions and aggregation functions). There is a parallel mapping between the graphs from graphical visual query language to the tree generated by Karma. However, Karma generates this tree automatically as users enter values, so they do not need to know the details of how sources and constraints are specified.

Programming by Demonstration[17, 35, 39] learns the procedure to perform a repetitive task by observing users perform a particular task. This approach can be effective in various domains[54, 36, 24] where users understand and know how to do such tasks. Once the procedure is learned, it can be executed to finish the task whenever it recognizes that the particular task is needed. In Karma, users may not know how to formulate queries and only interact with the system through data. In addition, the task of forming queries in Karma is unique, because every selection requires human judgment.

The mediator based system, such as Prometheus [57], uses planning techniques to translate a query, formulated according to the domain model, into multiple queries that integrate multiple sources. This approach is very powerful as the mediator system can detect overlapping conditions between sources and derive the optimal queries with the maximum data coverage. However, experts are required to use these systems since the mediator query must be formulated according to the predefined domain model.

Clio [65] is a system that supports user-based schema mapping and query building. The users can get suggestions about schema choices and join conditions. However, the system seems to focus on semi-expert users. While the users do not have to manually map schemas or write queries, they need to understand the source schema and database operations suggested by the system. Karma can create a more complex query because each tree that corresponds to each row of data in Karma can use different sources.

BANK [8] is a system that lets users create a query by typing in keywords that will be matched against current source relations and their attributes. BANK then creates multiple ranked queries and results. Q [56] is a similar system that improves over BANK by letting users provide feedback to help adjust the weight of each query. In Karma, the interface focuses on helping users generate the final data in the form of table. The user is shielded from complicated details of generated queries and data sources. On the other hand, Q focuses on helping users generate and refine queries though an interactive webform. It also allows users to fine tune a query by exposing which data sources are used.

Chapter 8

Evaluation

This chapter presents a formal user evaluation to validate my thesis statement and claims of research contributions. The overall evaluation plan follows the evaluation methodology outlined in [32]. First, I will state the claims and hypotheses that I want to evaluate. Second, I describe two types of users (i.e., programmers and non-programmers) in my evaluation and provide cretiria and justification for their selection. Third, based on those claims above, I explain three tasks designed to evaluate those claims. Fourth, I explain how the evaluation precedure is carried out in three steps: familiarization, practice, and test. Fifth, I outline the data collection and normalization process. Finally, I shows the evaluation result and discuses how it measures up to the claims made earlier.

8.1 Claims

1. Claim 1: User with no programming experiences can build all four Mashup types specified in section 2.1.3.

2. Claim 2: Karma takes less time to complete each subtask (e.g., data extraction, source modeling, data cleaning, and data integration) and scales better as the task gets more difficult.

   Rationale: Karma's approach allow users to specify operations by example, while the widget approach requires users to spend time customizing and verifying the result.

3. Claim 3: Overall, the user takes less time to build the same Mashup in Karma compared to Dapper/Pipes.

   Rationale: There are two factors that make Karma faster: a) Karma's approach of letting users specify results by example; and b.) Karma's idea of using information gained from one subproblem to help solve another subproblem.

8.2 Users

While the approach in this thesis is designed for users with no programming experience, the evaluation needs baseline tools (i.e., Karma and Dapper) for comparison. Given the time required to teach users how to use each of the systems, it was not possible to find nonprogrammer subjects to commit to the evaluation. As a trade off, there will be two types of users in my evaluations: users with programming experience and users with no programming experience. The user with programming experience will do each task outlined in section 8.3 twice: once using Dapper/Pipes and once using Karma. The user with no programming experience will do each task outlined in section 8.3 once using Karma. The evaluation result from nonprogrammers will be used to satisfy claim 1, while the result from the programmer will be used to satisfy claim 2 and claim 3. The background for each user type is discussed below.

8.2.1 Users with Programming Experience

These users are Master and PhD students in a graduate-level class focusing on the problem in information integration on the Web. The total number of students who participated in the evaluation is 20. They are prime candidates as subjects for the evaluation. Since they were given one assignment on Dapper and two assignments on Pipes in the course, they spent a significant amount of times learning and practicing with Dapper/Pipes.

8.2.2 Users with No Programming Experience

We have selected three users with no programming experience. One of the users is a master degree student in accounting and the others are administrative assistants. All of them has no prior programming experience, but they are familiar with Web technology and Excel.

8.3 Tasks

There will be three tasks in the evaluation. These tasks are designed to capture the structure of the four Mashup types discussed in section 2.1.3. The first task involves building a Mashup from one simple source. The second task is building a Mashup that combines multiple query results using database union from webpages with HTML form. Finally, the third task is building a Mashup that combines data from multiple sources using database join.

8.3.1 Baseline selection

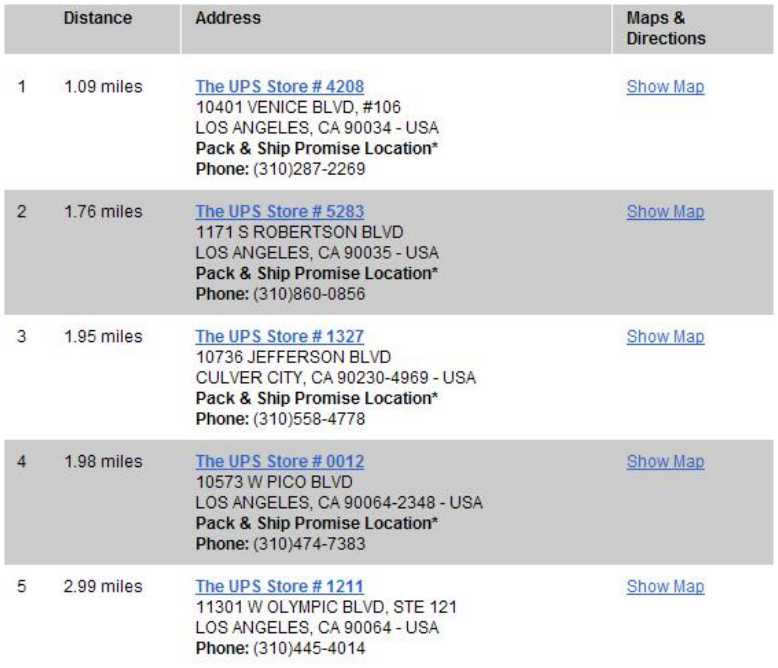
For each task, a programmer subject will build the same Mashup twice: once using Karma and once using the combination of Dapper and Pipes which we refer to as Dapper/Pipes and a nonprogrammer subject will build a Mashup for each task once. The result from Dapper/Pipes will be used as a comparison in terms of performance between the approach in my thesis and the widget approach.

As discussed in Chapter 7, Yahoo's Pipes is a state-of-the-art Mashup building system that employs the widget approach. Furthermore, it is readily available on the Web. As a result, I chose it as a baseline system for comparison. However, since Pipes does not have enough capability to do data extraction, I chose Dapper to fill that role; a subject has to extract data from a web page using Dapper, then process it using Pipes. This combination approach to building Mashups is often used by programmers who build Mashups.

8.3.2 Task 1: The UPS Store Locator

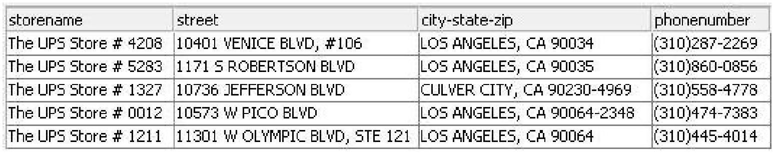
Figure 8.1 shows a snapshot of a website that lists UPS stores. In this task, a user needs to extract the following store information: store names, addresses, and phone numbers. This task maps to the first type of Mashup that only involves extracting data from one data source. In this Mashup type, the process usually involves data extraction, source modeling, and data cleaning. The data extraction part in this task is not trivial as there are optional tags in the DOM tree. For example, most entries have "*Pack & Ship Promise Location**", while one entry has no such data element.

Figure 8.2 shows the output that the user needs to generate in Karma, while Figure 8.3 shows the output that the user needs to generate in Pipes. Note that the output from



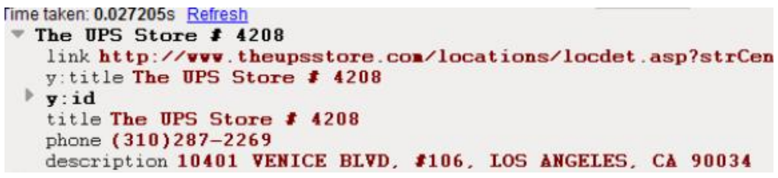
	Distance	Address	Maps & Directions
1	1.09 miles	The UPS Store # 4208 10401 VENICE BLVD, #106 LOS ANGELES, CA 90034 - USA **Pack & Ship Promise Location*** **Phone:** (310)287-2269	Show Map
2	1.76 miles	The UPS Store # 5283 1171 S ROBERTSON BLVD LOS ANGELES, CA 90035 - USA **Pack & Ship Promise Location*** **Phone:** (310)860-0856	Show Map
3	1.95 miles	The UPS Store # 1327 10736 JEFFERSON BLVD CULVER CITY, CA 90230-4969 - USA **Pack & Ship Promise Location*** **Phone:** (310)558-4778	Show Map
4	1.98 miles	The UPS Store # 0012 10573 W PICO BLVD LOS ANGELES, CA 90064-2348 - USA **Pack & Ship Promise Location*** **Phone:** (310)474-7383	Show Map
5	2.99 miles	The UPS Store # 1211 11301 W OLYMPIC BLVD, STE 121 LOS ANGELES, CA 90064 - USA **Phone:** (310)445-4014	Show Map

Figure 8.1: The first task involves extracting data from a listing of UPS stores.



storename	street	city-state-zip	phonenumber
The UPS Store # 4208	10401 VENICE BLVD, #106	LOS ANGELES, CA 90034	(310)287-2269
The UPS Store # 5283	1171 S ROBERTSON BLVD	LOS ANGELES, CA 90035	(310)860-0856
The UPS Store # 1327	10736 JEFFERSON BLVD	CULVER CITY, CA 90230-4969	(310)558-4778
The UPS Store # 0012	10573 W PICO BLVD	LOS ANGELES, CA 90064-2348	(310)474-7383
The UPS Store # 1211	11301 W OLYMPIC BLVD, STE 121	LOS ANGELES, CA 90064	(310)445-4014

Figure 8.2: The output that the user needs to generate in Karma for task 1.



```
Time taken: 0.027205s Refresh
▾ The UPS Store # 4208
    link http://www.theupsstore.com/locations/locdet.asp?strCen
    y:title The UPS Store # 4208
  ▸ y:id
    title The UPS Store # 4208
    phone (310)287-2269
    description 10401 VENICE BLVD, #106, LOS ANGELES, CA 90034
```

Figure 8.3: The output in RSS format that the user needs to generate in Pipes for task 1. While the RSS output for Dapper/Pipes has a built-in redundancy (i.e., multiple occurrence of titles), the user only needs to extract the data once.

Pipes is only in RSS format. As a result, the output shown in Figure 8.3 is the closest direct comparison to the output from Karma in Figure 8.2.

The data extracted can be put on a map and Karma has a capability to do so. However, since data display is not the focus of this thesis, the user is only asked to extract the data and put it in a cleaned format (i.e., street and city-state-zip). Note that the address part in the result specification contains no country (e.g., USA), so the user would need to clean the address part to get rid of that data token.

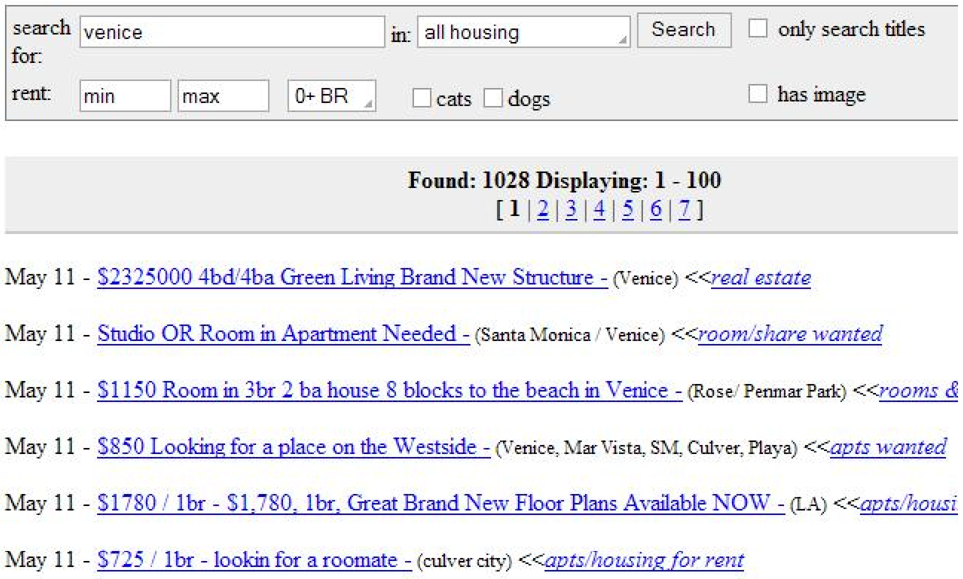
8.3.3 Task 2: Housing Rental Using Craigslist

Figure 8.4 shows a housing search section of a website called Craigslist. In this task, the user has to extract the post and its rental type from two areas: venice and westwood. To extract this information, the user needs to execute a form search in each area, extract the required data separately, and finally union it together. This task is a combination of the second and the third type of Mashup; database union and HTML form input.

Figure 8.5 shows the output that the user needs to generate in Karma, while Figure 8.6 shows the output that the user needs to generate in Pipes. Note that the post listing in the result specification contains no ending dash (-) symbol, so the user would need to clean each post to get rid of that character.

8.3.4 Task 3: Celebrity Address

Figure 8.7 shows a snapshot of a website with a list of actors. In this task, the user needs to extract the actors' names and link the names with another data source that contains their address, email, and URL information. This task maps to the forth type of Mashup



search for: venice in: all housing Search only search titles

rent: min max 0+ BR cats dogs has image

Found: 1028 Displaying: 1 - 100
[**1** | 2 | 3 | 4 | 5 | 6 | 7]

May 11 - $2325000 4bd/4ba Green Living Brand New Structure - (Venice) <<*real estate*

May 11 - Studio OR Room in Apartment Needed - (Santa Monica / Venice) <<*room/share wanted*

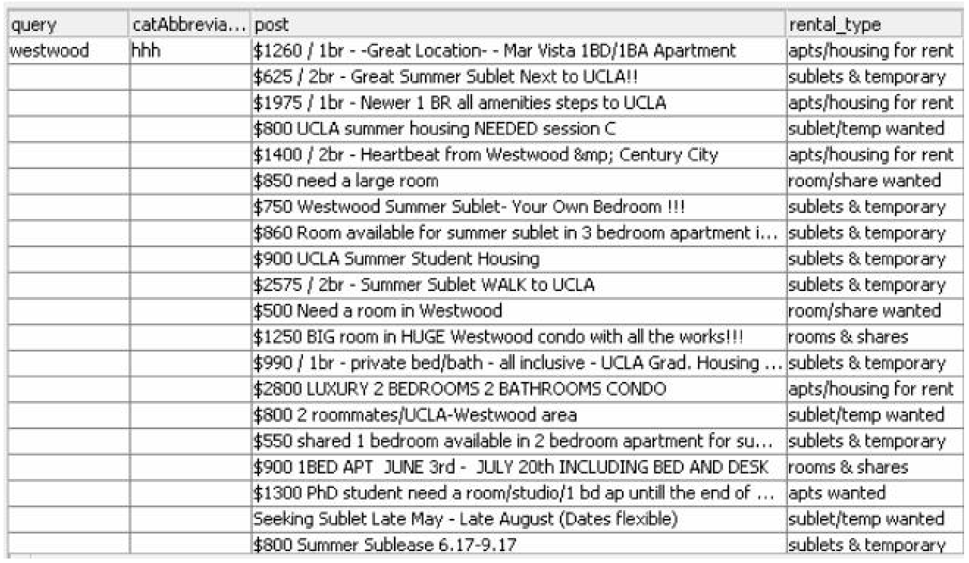
May 11 - $1150 Room in 3br 2 ba house 8 blocks to the beach in Venice - (Rose/ Penmar Park) <<*rooms &*

May 11 - $850 Looking for a place on the Westside - (Venice, Mar Vista, SM, Culver, Playa) <<*apts wanted*

May 11 - $1780 / 1br - $1,780, 1br, Great Brand New Floor Plans Available NOW - (LA) <<*apts/housi*

May 11 - $725 / 1br - lookin for a roomate - (culver city) <<*apts/housing for rent*

Figure 8.4: The second task involves extracting data from a website that has HTML form input.



query	catAbbrevia...	post	rental_type
westwood	hhh	$1260 / 1br - -Great Location- - Mar Vista 1BD/1BA Apartment	apts/housing for rent
		$625 / 2br - Great Summer Sublet Next to UCLA!!	sublets & temporary
		$1975 / 1br - Newer 1 BR all amenities steps to UCLA	apts/housing for rent
		$800 UCLA summer housing NEEDED session C	sublet/temp wanted
		$1400 / 2br - Heartbeat from Westwood &mp; Century City	apts/housing for rent
		$850 need a large room	room/share wanted
		$750 Westwood Summer Sublet- Your Own Bedroom !!!	sublets & temporary
		$860 Room available for summer sublet in 3 bedroom apartment i...	sublets & temporary
		$900 UCLA Summer Student Housing	sublets & temporary
		$2575 / 2br - Summer Sublet WALK to UCLA	sublets & temporary
		$500 Need a room in Westwood	room/share wanted
		$1250 BIG room in HUGE Westwood condo with all the works!!!	rooms & shares
		$990 / 1br - private bed/bath - all inclusive - UCLA Grad. Housing ...	sublets & temporary
		$2800 LUXURY 2 BEDROOMS 2 BATHROOMS CONDO	apts/housing for rent
		$800 2 roommates/UCLA-Westwood area	sublet/temp wanted
		$550 shared 1 bedroom available in 2 bedroom apartment for su...	sublets & temporary
		$900 1BED APT JUNE 3rd - JULY 20th INCLUDING BED AND DESK	rooms & shares
		$1300 PhD student need a room/studio/1 bd ap untill the end of ...	apts wanted
		Seeking Sublet Late May - Late August (Dates flexible)	sublet/temp wanted
		$800 Summer Sublease 6.17-9.17	sublets & temporary

Figure 8.5: The output that the user needs to generate in Karma for task 2. The Karma table contains the post and rental type, along with HTML form input information: query (westwood) and category abbreviation (hhh).

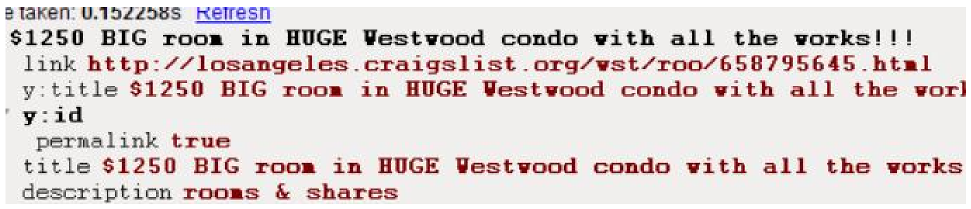
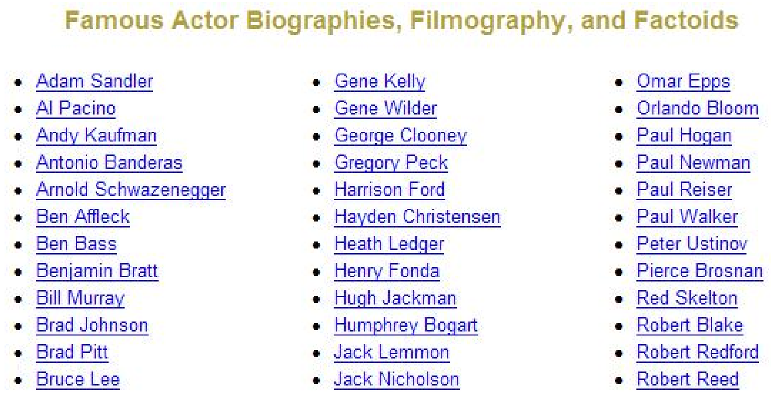

```
e taken: 0.152258s Refresh
$1250 BIG room in HUGE Westwood condo with all the works!!!
 link http://losangeles.craigslist.org/wst/roo/658795645.html
 y:title $1250 BIG room in HUGE Westwood condo with all the worl
 y:id
  permalink true
 title $1250 BIG room in HUGE Westwood condo with all the works
 description rooms & shares
```

Figure 8.6: The output in RSS format that the user needs to generate in Pipes for task 2. The rental type is stored under the description tag.

that combines two or more sources using a database join. In this case, the join key is the actor's name.

To simplify the running time of the source invocation with the second source that contains HTML form input, I have modified the web page to shorten the actor list, while preserving the structural integrity as shown in Figure 8.8.

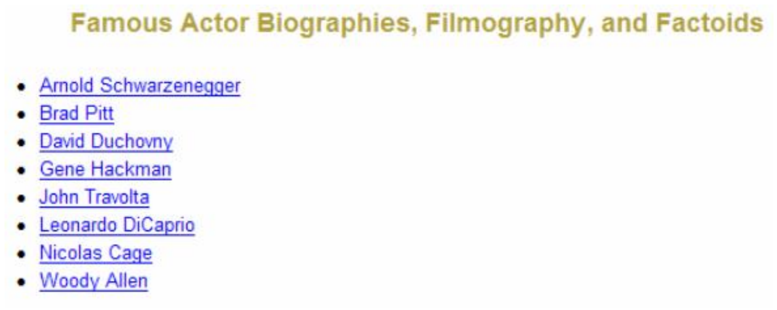
Figure 8.9 shows another web page that the subject could use to integrate with the actor page. This website (http://www.infospace.com/info/celeb/celebrity.htm) takes a celebrity name as an input and outputs his/her address, email, and URL information.


Famous Actor Biographies, Filmography, and Factoids

- Adam Sandler
- Al Pacino
- Andy Kaufman
- Antonio Banderas
- Arnold Schwazenegger
- Ben Affleck
- Ben Bass
- Benjamin Bratt
- Bill Murray
- Brad Johnson
- Brad Pitt
- Bruce Lee
- Gene Kelly
- Gene Wilder
- George Clooney
- Gregory Peck
- Harrison Ford
- Hayden Christensen
- Heath Ledger
- Henry Fonda
- Hugh Jackman
- Humphrey Bogart
- Jack Lemmon
- Jack Nicholson
- Omar Epps
- Orlando Bloom
- Paul Hogan
- Paul Newman
- Paul Reiser
- Paul Walker
- Peter Ustinov
- Pierce Brosnan
- Red Skelton
- Robert Blake
- Robert Redford
- Robert Reed

Figure 8.7: The original web page containing a list of actors.

This task demonstrates the difficulty of finding the right data sources to combine. In this task, the user will be given information only about the first website (i.e., a list of actors). For the Dapper/Pipes task, the user would have to search the previously built



Famous Actor Biographies, Filmography, and Factoids

- Arnold Schwarzenegger
- Brad Pitt
- David Duchovny
- Gene Hackman
- John Travolta
- Leonardo DiCaprio
- Nicolas Cage
- Woody Allen

Figure 8.8: The modified web page containing a shortened list of actors.

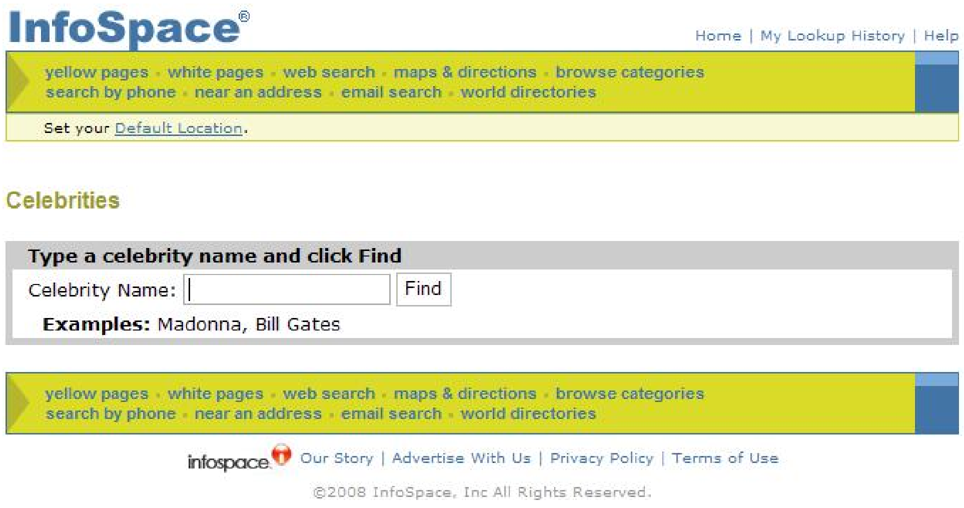
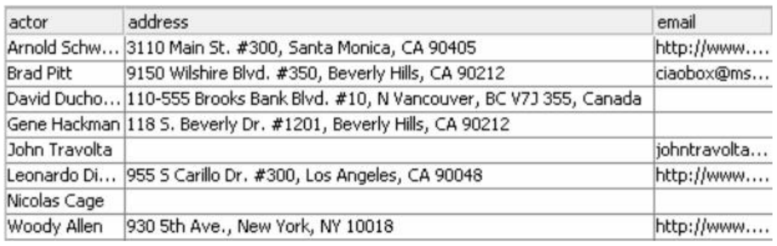

InfoSpace®
Home | My Lookup History | Help
yellow pages · white pages · web search · maps & directions · browse categories
search by phone · near an address · email search · world directories
Set your Default Location.
Celebrities
Type a celebrity name and click Find
Celebrity Name:
Find
Examples: Madonna, Bill Gates
yellow pages · white pages · web search · maps & directions · browse categories
search by phone · near an address · email search · world directories
infospace Our Story | Advertise With Us | Privacy Policy | Terms of Use
©2008 InfoSpace, Inc All Rights Reserved.

Figure 8.9: The second data source used for joining with the first data source.



actor	address	email
Arnold Schw...	3110 Main St. #300, Santa Monica, CA 90405	http://www....
Brad Pitt	9150 Wilshire Blvd. #350, Beverly Hills, CA 90212	ciaobox@ms...
David Ducho...	110-555 Brooks Bank Blvd. #10, N Vancouver, BC V7J 3S5, Canada	
Gene Hackman	118 S. Beverly Dr. #1201, Beverly Hills, CA 90212	
John Travolta		johntravolta...
Leonardo Di...	955 S Carillo Dr. #300, Los Angeles, CA 90048	http://www....
Nicolas Cage		
Woody Allen	930 5th Ave., New York, NY 10018	http://www....

Figure 8.10: The output that the user needs to generate in Karma for task 3.

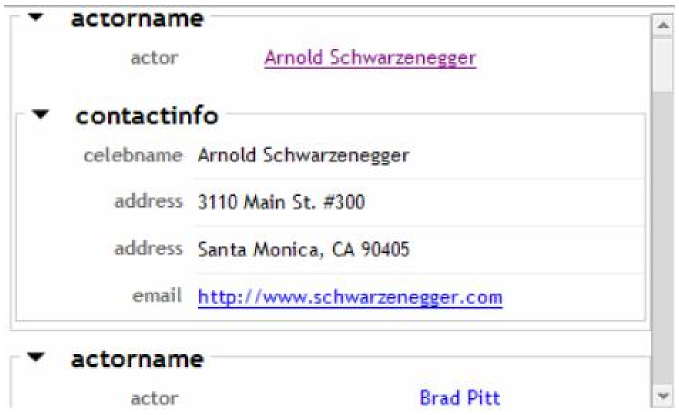

actorname
actor Arnold Schwarzenegger
contactinfo
celebname Arnold Schwarzenegger
address 3110 Main St. #300
address Santa Monica, CA 90405
email http://www.schwarzenegger.com
actorname
actor Brad Pitt

Figure 8.11: The output that the user needs to generate in Dapper/Pipes for task 3. Since Dapper offers a join operator, while Pipes only offers a union operator, this task is done solely using Dapper and the output is displayed using Dapper.

Infospace celebrity Dapp to integrate with the actor source. For the Karma task, the user would have to use the data integration feature to locate attributes that can be expanded based on the current data extracted from a new source.

Figure 8.10 shows the output that the user needs to generate in Karma, while Figure 8.11 shows the output that the user needs to generate in Dapper. Specifically, the result in Karma is generated by combining the data from the first data source (obtained from extraction) with the infospace celebrity data source (stored in Karma's database as one of the tables). The result in Dapper is generated by linking the data from the first Dapp (obtained by wrapping the first web source) with the second Dapp (founded by searching through a list of previously created Dapps by other users).

8.3.5 Difficulty Breakdown for Each Mashup Building Problem

Table 8.1 show how difficult it is to solve each Mashup building subtask in each task. Differentiating how hard each subtask is will allow me to do fine grain comparisons for each subtask as well as overall comparisons.

Table 8.1: Difficulty breakdown for each Mashup building subtask in each task. Note that ask 1 has no data integration subtask, while task 3 has no data cleaning subtask

	Data Extraction	Source Modeling	Data Cleaning	Data Integration
Task 1	Moderate	Simple	Hard	N/A
Task 2	Hard	Simple	Simple	Union (Simple)
Task 3	Simple	Simple	N/A	Join (Hard)

In terms of data extraction subtasks, the first task is moderately difficult because the DOM structure is irregular. The second task is difficult because it involves extracting data from a site with an HTML form.

In terms of source modeling subtasks, they are all simple, because a) Dapper/Pipes allows users to type in any attribute name they want; and b) Karma will suggest attributes automatically for the user if there is an overlapping set of values in the database (as discussed in Chapter 4).

In terms of data cleaning subtasks, the first task is difficult because the irregular DOM results in poor extracted data. The data cleaning in the second task is simple, as it involves removing a dash symbol at the end of each post. Task 3 does not have any data cleaning subtask.

In terms of data integration subtasks, task 1 is not applicable as no data integration is required. Task 2 requires a database union, while task 3 requires a database join. As a result, task 3 is more difficult than task 2, because specifying a join condition is more complicated.

8.4 Experimental Procedure

The experimental procedure is divided into three phrases: familiarization, practice, and test. The detail for each phrase is discussed below.

8.4.1 Familiarization

For subjects with programming experience, the tutorial for Dapper, Pipes, and Karma (videos and Word documents) is sent out two days before the experiment date. The tutorial contains all the concepts and examples of Mashups structurally similar to the three evaluation tasks. On the day of the experiment, each subject is given a quick 30 minute tutorial that covers all three systems to refresh their memory. For subjects with no programming experience, I gave a 15 minute tutorial on Karma on the day of the experiment.

8.4.2 Practice

After finishing the quick tutorial, both type of subjects are given two practice tasks in Karma. Note that these subjects have seen a tutorial on Karma but have never used Karma. The two practice tasks cover the basics of how to extract data, assign attributes, clean data, and integrate data across multiple sources. These two practice tasks take about 10 minutes to complete and give them a chance to use Karma.

8.4.3 Test

The detail of the test phrase is listed below:

- The test phrase lasts about one hour.

- Each subject is given three tasks discussed in section 8.3. The order of tasks given is task 3, task 2, and task 1.
- For subjects with programming experience, the subject starts implementing each task in Karma first. Then the subject does the same task using Dapper/Pipes. Note that this setup gives an advantage to Dapper/Pipes, because the subject can familiarize herself with web sources and requirements while finishing the task using Karma.
- For subjects with no programming experience, the subject starts implementing each task only in Karma.
- Hints and advice are given when asked.
- The computer screen is recorded using the video capture software for later analysis.
- If a subject gets stuck in a particular subtask (e.g., cleaning and integration) more than 5 minutes, the cutoff time is enforced and task will be marked as *fail*.
- If the subject makes a mistake that crashes the software, the software will be restarted and the subject will have to redo all previous steps again. However, the time used for the analysis will be the time taken prior to that mistake plus the time taken to finish the task after the subject redoes all the previous steps up to the point of the mistake.

8.5 Collecting and Processing Data

As mentioned earlier, the computer screen is recorded while subjects are completing three evaluation tasks. These video records allow us to see how long each subject took to complete each task and what kind of choices and options each subject made.

8.5.1 Normalizing Data

All the videos recorded are segmented into multiple time slots based on the four subtasks (i.e., data extraction, source modeling, data cleaning, and data integration) and other miscellaneous operations. Table 8.2 shows an example how the raw data is first processed.

Table 8.2: An example showing how the data is segmented for subject No.1 doing the second task using Dapper/Pipes.

Subtask	Time Range (in minutes)	Time Taken (in minutes)
Data Extraction	0:00 - 2:10	2:10
Page Loading	2:10 - 2:22	0:12
Data Extraction	2:22 - 2:50	0:28
Source Modeling	2:50 - 3:12	0:22
Page Loading	3:12 - 3:30	0:18
Data Extraction	3:30 - 5:30	2:00
Data Integration	5:30 - 6:45	1:15
Data Cleaning	6:45 - 9:30	2:45

To ensure fairness in the evaluation, the following segments are not counted toward the time taken: page loading time, and interfacing time between Dapper and Pipes (when applicable). During the experiment, if the time allocated for the whole evaluation (one hour) is up and a subject does not have time to start working on a particular task, the data is marked as *n/a*. On the other hand, if a subject cannot complete a task because one or more of its subtasks are too difficult, then the time spent on that particular subtask is marked as *Fail*.

To enable a comparison when a subject fails on a task, there needs to be a way to quantify those failures. In the final result, we substitute the *Fail* slot with the cut off time of 5 minutes.

8.5.2 Experimental Results

After normalizing the data, the final results in terms of minutes spent, segmented by subtasks, in each task are shown in table 8.3, 8.4, and 8.5. Note that since there are two subjects (i.e., No.3 and No.17) who did not have time to finish task 1, the result from these two subjects will be excluded from results that involve task 1 to ensure a fair comparison.

8.6 Discussion

In this section, we discuss the results from the experiment. The results will be segmented and discussed based on each of the claims made earlier. For claim 2 and claim 3, we compute statistical significance tests to support our results. In our evaluation, we imposed a time limit of 5 minutes for each subtask. Note that [53] argued that time bound experiment can bias the result of the evalution and Etzioni [22] extensively discussed some possible solutions. In our case, the time bound is applied to both Dapper/Pipe and Karma result. However, every subject can finish his/her task under the time limit using Karma. As a result, the scenario introduced in [53] that shows a possible bias in time bound experiments does not apply to our experimental results.

Table 8.3: Normalized data for task. E stands for data extraction, M stands for source modeling, and C stands for data cleaning. The asterisk indicates time substitution when failures happen. The data is reported in minutes. The first 20 subjects have programming background, while the last 3 subjects have no programming background.

Task1	Dapper/Pipes				Karma			
Subject	E	M	C	Total	E	M	C	Total
No.1	*5:00	0:20	*5:00	10:20	2:19	1:08	1:00	4:27
No.2	1:43	0:30	*5:00	7:13	1:00	0:40	0:29	2:09
No.3	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
No.4	0:52	0:48	*5:00	6:40	1:12	1:00	0:50	3:02
No.5	5:00	0:35	*5:00	10:35	1:15	1:18	1:20	3:53
No.6	2:30	0:15	*5:00	7:45	1:00	0:54	0:28	2:22
No.7	1:20	0:22	*5:00	6:42	0:51	0:51	0:46	2:28
No.8	1:40	0:14	*5:00	6:54	1:04	0:41	0:33	2:19
No.9	1:26	0:16	*5:00	6:42	1:14	1:00	1:10	3:24
No.10	1:39	0:10	*5:00	6:49	0:53	0:42	0:50	2:26
No.11	2:00	0:19	*5:00	7:19	1:04	1:00	0:53	2:57
No.12	2:00	0:49	2:00	4:49	1:07	1:00	0:40	2:47
No.13	2:00	0:05	*5:00	7:05	0:58	0:50	0:56	1:44
No.14	2:46	0:15	*5:00	8:01	1:12	0:45	0:48	2:45
No.15	2:27	0:14	3:11	5:52	1:10	0:49	1:20	3:19
No.16	1:16	0:12	*5:00	6:28	0:58	0:42	0:25	2:05
No.17	n/a	n/a	n/a	n/a	2:00	1:00	0:50	3:50
No.18	2:30	0:14	*5:00	7:44	1:06	1:10	1:46	4:02
No.19	1:38	0:47	1:20	3:45	1:20	0:49	0:35	2:44
No.20	1:30	0:16	*5:00	6:46	1:04	0:44	0:35	2:23
No.21	n/a	n/a	n/a	n/a	1:11	1:17	0:59	3:27
No.22	n/a	n/a	n/a	n/a	2:58	1:46	1:24	6:08
No.23	n/a	n/a	n/a	n/a	1:19	1:40	1:14	4:13

Table 8.4: Normalized data for task 2. E stands for data extraction, M stands for source modeling, C stands for data cleaning, and I stands for data integration. The asterisk indicates time substitution when failures happen. The data is reported in minutes. The first 20 subjects have programming background, while the last 3 subjects have no programming background.

Task2	Dapper/Pipes					Karma				
Subject	E	M	C	I	Total	E	M	C	I	Total
No.1	4:38	0:22	2:45	1:15	9:00	1:26	0:43	0:43	0:00	2:52
No.2	1:35	0:12	3:30	0:12	5:29	0:50	0:57	0:57	0:00	2:44
No.3	*5:00	0:25	*5:00	*5:00	15:25	2:52	1:00	3:00	0:00	5:52
No.4	4:49	0:17	3:29	0:38	9:14	1:26	0:48	1:03	0:00	3:18
No.5	*5:00	0:29	1:44	1:16	8:29	1:43	0:45	1:20	0:00	3:48
No.6	*5:00	0:20	*5:00	*5:00	15:20	2:07	0:30	0:50	0:00	3:27
No.7	2:17	0:15	4:46	0:18	7:36	1:13	0:25	0:52	0:00	2:31
No.8	3:23	0:21	*5:00	*5:00	13:44	1:10	0:21	0:24	0:00	1:55
No.9	4:11	0:21	*5:00	*5:00	14:32	1:22	0:47	2:11	0:00	4:20
No.10	2:16	0:07	3:14	0:20	5:50	1:04	0:20	1:06	0:00	2:30
No.11	3:04	0:17	*5:00	*5:00	13:21	1:06	0:34	0:53	0:00	2:33
No.12	2:00	0:27	*5:00	0:20	7:47	1:23	0:30	0:37	0:00	2:30
No.13	*5:00	0:07	1:43	0:10	7:00	1:42	0:32	0:41	0:00	2:55
No.14	3:03	0:23	4:42	0:10	8:21	1:40	0:31	0:56	0:00	3:07
No.15	2:06	0:12	3:13	0:22	5:53	1:30	0:24	2:05	0:00	3:59
No.16	3:58	0:11	3:29	0:27	8:05	0:51	0:17	1:00	0:00	2:08
No.17	4:15	0:28	3:39	0:30	8:52	1:04	0:28	1:18	0:00	2:50
No.18	*5:00	0:23	*5:00	*5:00	15:23	1:17	0:30	1:10	0:00	2:57
No.19	4:01	0:14	2:42	0:21	7:16	1:39	0:21	0:50	0:00	2:50
No.20	1:36	0:43	0:36	0:22	3:17	1:07	0:28	0:40	0:00	2:15
No.21	n/a	n/a	n/a	n/a	n/a	1:03	0:21	0:55	0:00	2:19
No.22	n/a	n/a	n/a	n/a	n/a	3:56	1:52	2:50	0:00	8:38
No.23	n/a	n/a	n/a	n/a	n/a	2:15	0:31	1:27	0:00	4:13

Table 8.5: Normalized data for task 3. E stands for data extraction, M stands for source modeling, and I stands for data integration. The asterisk indicates time substitution when failures happen. The data is reported in minutes. The first 20 subjects have programming background, while the last 3 subjects have no programming background.

Task3	Dapper/Pipes				Karma			
Subject	E	M	I	Total	E	M	I	Total
No.1	1:30	0:26	*5:00	6:56	0:14	0:00	2:16	2:30
No.2	0:30	0:10	*5:00	5:40	0:25	0:00	0:26	0:54
No.3	1:00	0:15	*5:00	6:15	0:15	0:00	0:44	0:59
No.4	0:40	0:16	*5:00	5:56	0:20	0:00	1:06	1:26
No.5	0:40	0:14	*5:00	5:54	0:20	0:00	0:37	0:57
No.6	0:30	0:10	*5:00	5:40	0:20	0:00	0:31	0:51
No.7	0:27	0:10	*5:00	5:37	0:14	0:00	0:50	1:04
No.8	0:29	0:20	*5:00	5:49	0:30	0:00	0:51	1:21
No.9	0:40	0:23	*5:00	6:03	0:13	0:00	0:44	0:57
No.10	0:30	0:10	*5:00	5:40	0:20	0:00	0:35	0:55
No.11	0:51	0:20	*5:00	6:11	0:16	0:00	1:05	1:21
No.12	1:05	0:18	*5:00	6:23	0:30	0:00	0:46	1:16
No.13	0:31	0:14	*5:00	5:45	0:16	0:00	0:57	1:13
No.14	0:36	0:14	*5:00	5:50	0:14	0:00	2:00	2:14
No.15	0:26	0:21	*5:00	5:47	0:30	0:00	0:45	1:15
No.16	0:27	0:13	*5:00	5:40	0:15	0:00	0:56	1:11
No.17	0:33	0:38	1:56	3:07	0:30	0:00	0:46	1:16
No.18	1:03	0:07	*5:00	6:10	0:20	0:00	1:10	1:30
No.19	0:33	0:17	*5:00	5:50	0:25	0:00	1:20	1:45
No.20	0:18	0:13	*5:00	5:31	0:12	0:00	0:44	0:56
No.21	n/a	n/a	n/a	n/a	0:15	0:00	0:39	0:54
No.22	n/a	n/a	n/a	n/a	0:21	0:00	2:50	3:11
No.23	n/a	n/a	n/a	n/a	0:12	0:00	0:51	1:03

8.6.1 Nonprogrammers vs Programmer

Note that the evaluation results from nonprogrammer are used to satisfy claim 1, while the result from programmers is used to satisfy claim 2 and claim 3. We believe that if programmer subjects, who are familiar with workflows and widget paradigms in general, spend more time to implement a task using Dapper/Pipes compared to Karma, then nonprogrammer subjects would also spend more time to implement the same task using Dapper/Pipes (if they were to learn how to use these systems) compared to Karma.

8.6.2 Claim 1: User with no programming experiences can build all four Mashup types

Figure 8.12 shows how much time is spent in each of the tasks for each nonprogrammer subject. Overall, each subject was able to complete all three tasks (designed to be representative of four Mashup types) without failing. On average, it takes nonprogrammers 3:47 minutes to build a Mashup.

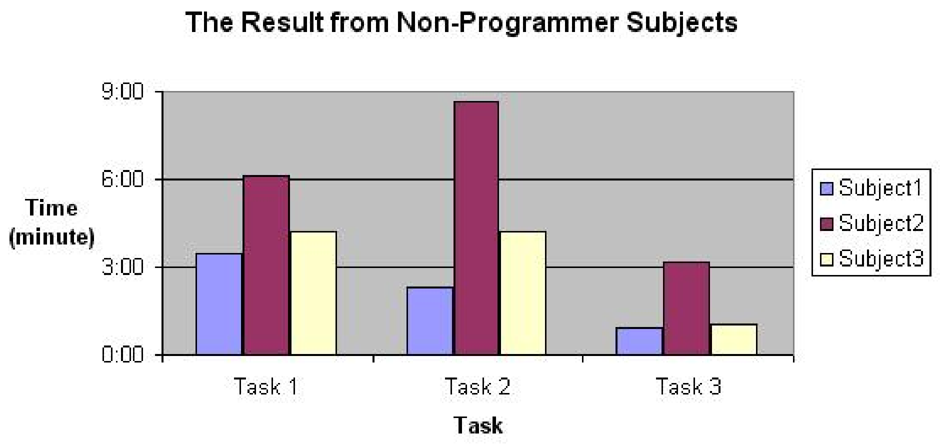


Figure 8.12: The result from nonprogrammer evaluation for Karma segmented by tasks.

Table 8.3, 8.4, and 8.5 show the detailed results of how each nonprogrammer (i.e., subject No. 21-23) performs on each task using Karma and confirm our claim that user with no programming experiences can build all four Mashup types.

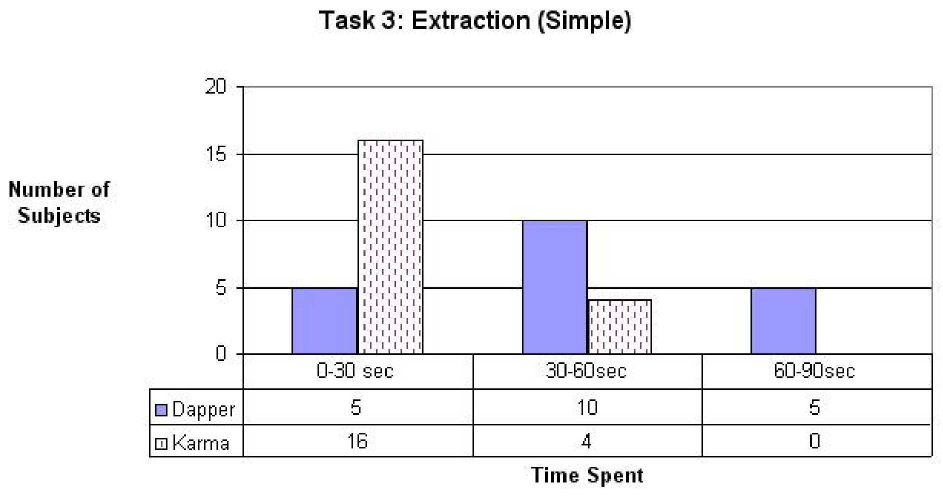
8.6.3 Claim 2: Karma takes less time to complete each subtask and scales better as the task gets more difficult.

In this subsection, we will show the comparison for each subtask based on its difficulty. Specifically, the result from data extraction (Figure 8.13, 8.14, 8.15), data cleaning (Figure 8.19, 8.20), and data integration (Figure 8.21, 8.22) all show that when a subtask gets harder, subjects take more time to complete that subtask using Dapper/Pipes compared to Karma. Note that while the results show that Karma performs worse in source modeling (Figure 8.16 and Figure 8.17), the time spent during the source modeling helps subjects finish the Mashup task faster during the integration subtask (Figure 8.22).

8.6.3.1 Data Extraction

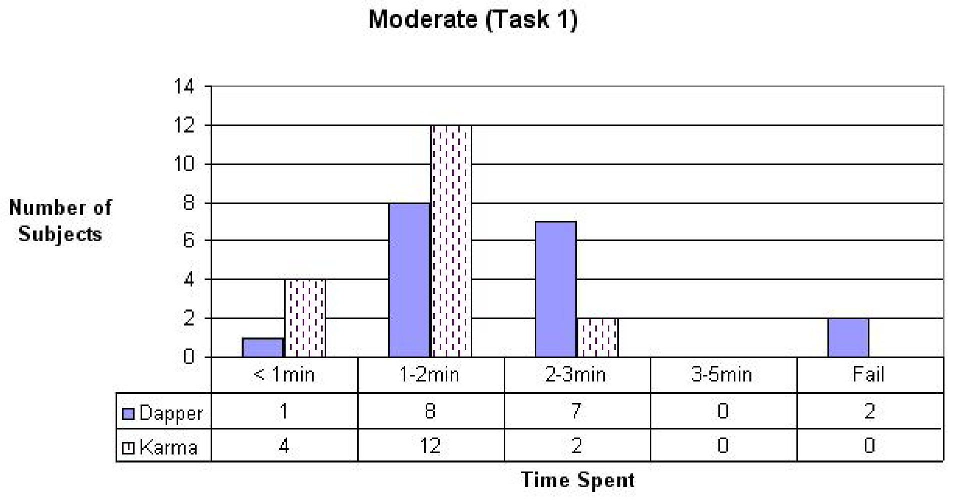
Figures 8.13, 8.14, and 8.15 show the performance measurement between Karma and Dapper/Pipes for the data extraction subtask. In each graph, the x-axis is the time spent to complete the data extraction subtask, while the y-axis is the number of subjects that fall into each time segment.

In task 3, the extraction task is very simple as it involves extracting only one field of data. The result shown in Figure 8.13 indicates that most subjects can finish the extraction task using Karma in less than 30 seconds, while most of them finish the extraction task using Dapper in around 30 seconds to one minute.



	0-30 sec	30-60sec	60-90sec
▭ Dapper	5	10	5
▨ Karma	16	4	0

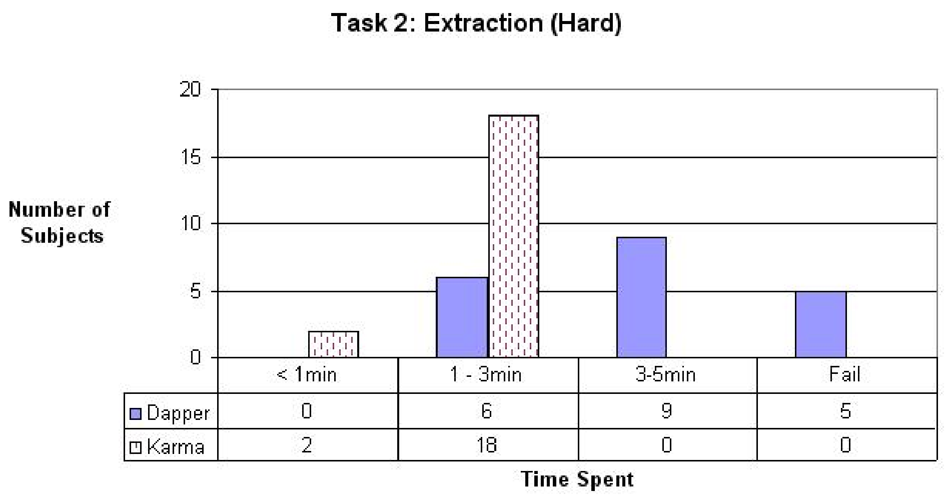
Figure 8.13: The performance comparison (programmers) between Karma and Dapper/Pipes for the data extraction subtask (simple) in Task 3.



	< 1min	1-2min	2-3min	3-5min	Fail
▭ Dapper	1	8	7	0	2
▨ Karma	4	12	2	0	0

Figure 8.14: The performance comparison (programmers) between Karma and Dapper/Pipes for the data extraction subtask (moderate difficulty) in Task 1.

In task 1, the extraction task is of medium difficulty because of the irregular DOM structure of the web source. The result shown in Figure 8.14 shows that Karma performs better as more subjects finish the subtask faster using Karma.



	< 1min	1 - 3min	3-5min	Fail
▪ Dapper	0	6	9	5
▫ Karma	2	18	0	0

Figure 8.15: The performance comparison (programmers) between Karma and Dapper/Pipes for the data extraction subtask (hard) in Task 2.

In task 2, the extraction task is hard because it involves extracting data from a data source with HTML form. The result shown in Figure 8.15 shows that Karma performs better than Dapper/Pipes as all subjects finish the subtask using Karma in less than 3 minutes, while 9 subjects take 3 minutes and longer and 14 subject fail to finish the same subtask using Dapper/Pipes.

Table 8.6 shows the T-test for the hypothesis that Karma is faster than Dapper/Pipes for data extraction in each of the tasks and the overall subtask.

We also want to show that as the extraction task gets progressively harder, Karma performs better in terms of time spent to finish the subtask. For each programmer subject, we compute the following values:

Table 8.6: The statistical significance test result for data extraction subtask

Task No	T-test
Task 3 (Simple)	t=4.69, degree of freedom=38, and $p < 0.01$
Task 1 (Moderate)	t=3.58, degree of freedom=34, and $p < 0.01$
Task 2 (Hard)	t=7.05, degree of freedom=38, and $p < 0.01$
Overall data extraction	t=5.35, degree of freedom=114, and $p < 0.01$

1. DiffD: $\Sigma\{$time spent using Dapper/Pipes in task 2 (extraction, hard) - time spent using Dapper/Pipes in task 3 (extraction, easy)$\}$

2. DiffK: $\Sigma\{$time spent using Karma in task 2 (extraction, hard) - time spent using Karma in task 3 (extraction, easy)$\}$

Table 8.7: The comparision between DiffD and DiffK for each programmer subject for data extraction subtasks. The unit is in seconds.

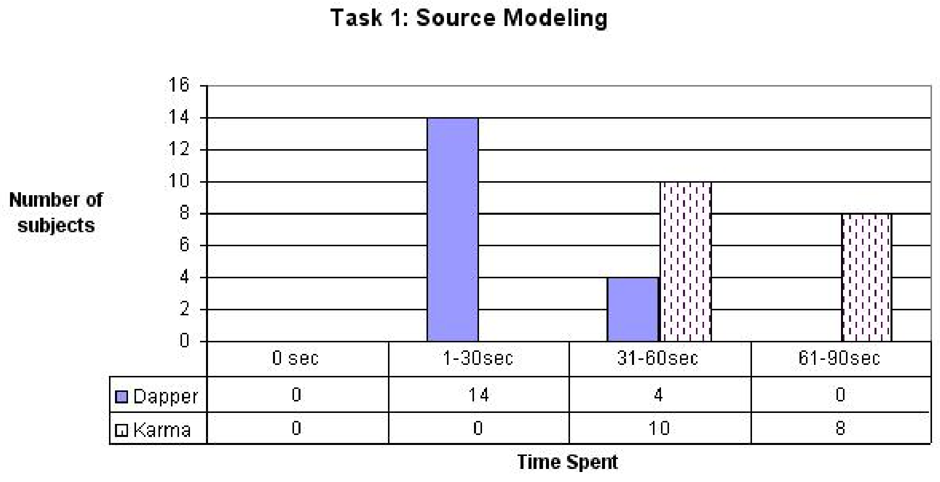
Subject No.	DiffD	DiffK
No.1	188	72
No.2	65	25
No.4	249	66
No.5	260	83
No.6	271	107
No.7	110	59
No.8	174	40
No.9	211	69
No.10	106	44
No.11	133	50
No.12	55	53
No.13	269	86
No.14	147	86
No.15	100	60
No.16	211	36
No.18	237	57
No.19	208	74
No.20	78	55
Average	170	62

Table 8.7 show the time comparison of DiffD and DiffK. For example, the subjects spent, on average, 170 seconds more to finish the difficult extraction subtask in task 2

compared to the easy extraction subtask in task 3, when using Dapper/Pipes. On the other hand, the same group spent 62 seconds, on average, more to finish the difficult extraction subtask in task 2 compared to the easy extraction subtask in task 3, when using Karma. What this table shows is that as the extraction subtask gets more difficult, the increment in time spent using Dapper/Pipes is more than the increment in time spent using Karma. In addition, the result is statistically significant (t=5.87, degree of freedom=34, and $p < 0.01$)

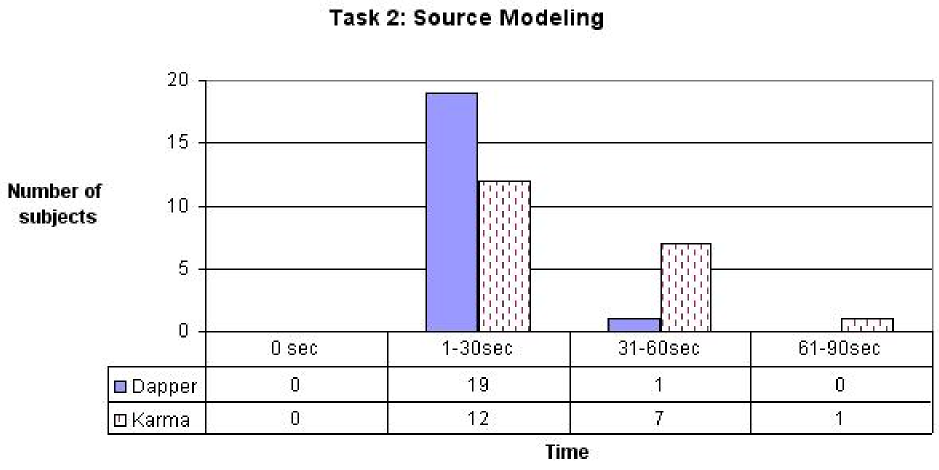
8.6.3.2 Source Modeling

Figures 8.16, 8.17, and 8.18 show the performance measurement between Karma and Dapper/Pipes for the source modeling subtask. In each graph, the x-axis is the time spent to complete the data extraction subtask, while the y-axis is the number of subjects that fall into each time segment.



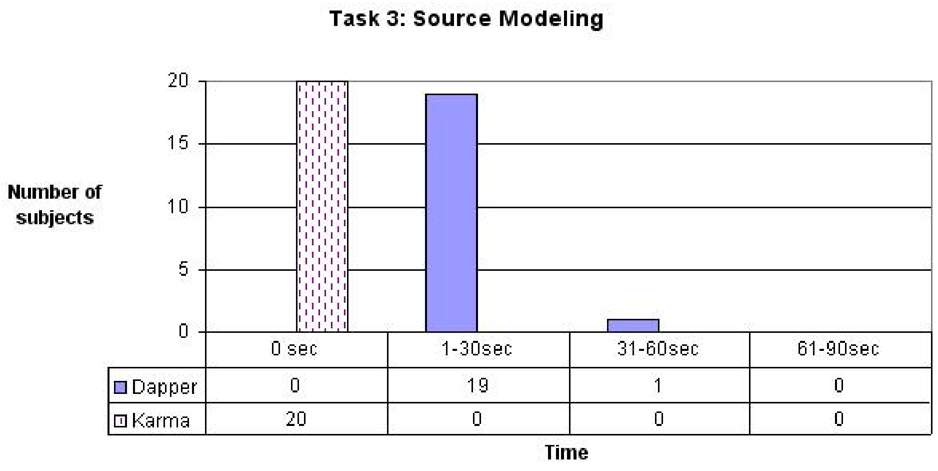
	0 sec	1-30sec	31-60sec	61-90sec
▨ Dapper	0	14	4	0
▢ Karma	0	0	10	8

Figure 8.16: The performance comparison (programmers) between Karma and Dapper/Pipes for the source modeling subtask in Task 1.



	0 sec	1-30sec	31-60sec	61-90sec
■ Dapper	0	19	1	0
▨ Karma	0	12	7	1

Figure 8.17: The performance comparison (programmers) between Karma and Dapper/Pipes for the source modeling subtask in Task 2.



	0 sec	1-30sec	31-60sec	61-90sec
■ Dapper	0	19	1	0
▨ Karma	20	0	0	0

Figure 8.18: The performance comparison (programmers) between Karma and Dapper/Pipes for the source modeling subtask in Task 3.

The results from Figure 8.16 and 8.17 suggest that Dapper/Pipes is faster than Karma in task 1 and task 2. Karma does perform better in task 3 where attributes are set automatically because of value overlapping in the database. However, Karma performs worse than Dapper/Pipes overall, because of two factors:

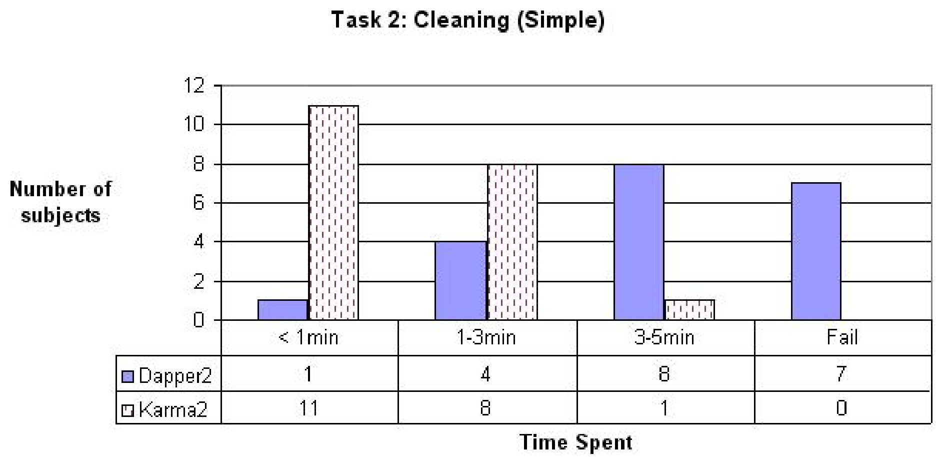
- The table implementation of Java in Karma does not allow a user to set the attribute directly by clicking at the attribute cell. Users need to go to the attribute tab to select an attribute name for each column.
- When using Karma, subjects also look to see what kind of attributes are suggested by Karma, while they can just type in any attribute value for Dapper.

While Karma performs worse than Dapper/Pipes in this subtask, notice that the time spent doing source modeling compared to the overall Mashup building task is small. Furthermore, by assigning the right attribute using Karma, it will be appararent that users will save more time during the data integration subtask that involves a database join (task 3).

8.6.3.3 Data Cleaning

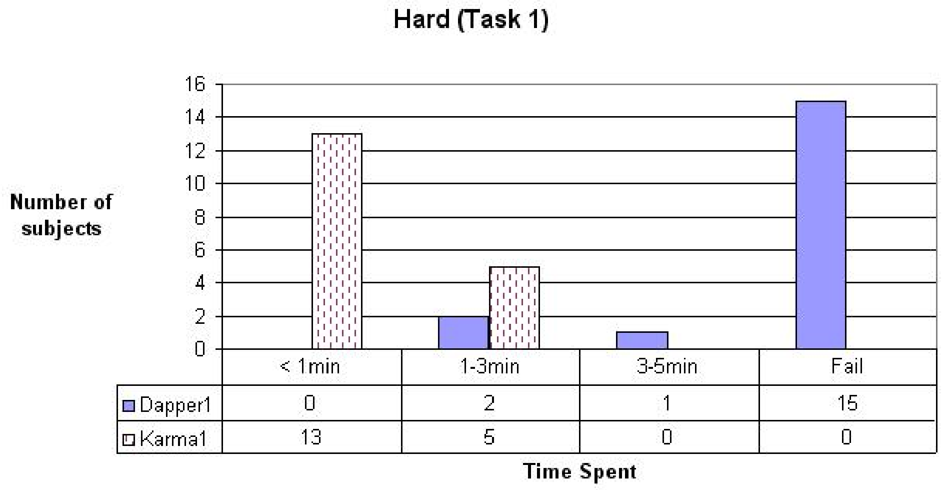
Figures 8.19 and 8.20 show the performance comparison between Karma and Dapper/Pipes for the data cleaning subtask. In each graph, the x-axis is the time spent to complete the data cleaning subtask, while the y-axis is the number of subjects that fall into each time segment.

Table 8.8 shows the T-test for the hypothesis that Karma is faster than Dapper/Pipes for data cleaning in each of the task and the overall subtask. We also want to show that



	< 1min	1-3min	3-5min	Fail
▨ Dapper2	1	4	8	7
□ Karma2	11	8	1	0

Figure 8.19: The performance comparison (programmers) between Karma and Dapper/Pipes for the data cleaning subtask (simple) in Task 2.



	< 1min	1-3min	3-5min	Fail
▨ Dapper1	0	2	1	15
□ Karma1	13	5	0	0

Figure 8.20: The performance comparison (programmers) between Karma and Dapper/Pipes for the data cleaning subtask (hard) in Task 1.

as the data cleaning subtask gets progressively harder, Karma performs better in terms of time spent to finish the subtask. Table 8.9 shows the difference between **DiffD** and **DiffK** for the data cleaning subtask. The data in table 8.9 differs from that of table 8.7 because it contains negative numbers and zeros. For example, DiffK for subject No.2 is -28, which means that as the task gets harder, the subject actually spent less time solving it using Karma.

Table 8.8: The statistical significance test result for data cleaning subtask

Task No	T-test
Task 2 (Simple)	t=7.68, degree of freedom=38, and $p < 0.01$
Task 1 (Hard)	t=12.76, degree of freedom=34, and $p < 0.01$
Overall data cleaning	t=13.54, degree of freedom=74, and $p < 0.01$

Table 8.9: The comparision between DiffD and DiffK for each programmer subject for data cleaning subtasks. The unit is in seconds.

Subject No.	DiffD	DiffK
No.1	135	17
No.2	90	-28
No.4	91	-13
No.5	196	0
~~No.6~~	~~0~~	~~-22~~
No.7	14	-6
~~No.8~~	~~0~~	~~9~~
~~No.9~~	~~0~~	~~-61~~
No.10	106	-16
~~No.11~~	~~0~~	~~0~~
No.12	-180	3
No.13	197	15
No.14	18	-8
No.15	-2	-45
No.16	91	-35
~~No.18~~	~~0~~	~~36~~
No.19	-82	-15
No.20	264	-5
Average	72	-10

The value zero in **DiffD** is the result from a subject failing to complete both cleaning subtasks with the time bound. Since we use the time bound of five minutes, **DiffD** is 0. Note that this is a case of doubly censored data [22]. In the case of doubly censored data, where it is expensive to obtain more sample points, the standard statistical practice is to throw away such data and consider the data that has a) singly censored pairs or b) uncensored pairs [63]. As a result, the results from subject No.6,8,9,11,18 are discarded and they are not factored in the mean. We found that the increment in time spent using Dapper/Pipes is more than the increment in time spent using Karma for data cleaning subtasks. In addition, the result is statistically significant (t=1.96, degree of freedom=24, and $p < 0.05$)

The performance comparison difference between Karma and Dapper/Pipes is more obvious in the data cleaning subtask compared to the earlier two subtasks. Furthermore, only about 40 percents of subjects can complete the data cleaning subtask in task 2 using Pipes, while all subjects can complete the same subtask using Karma.

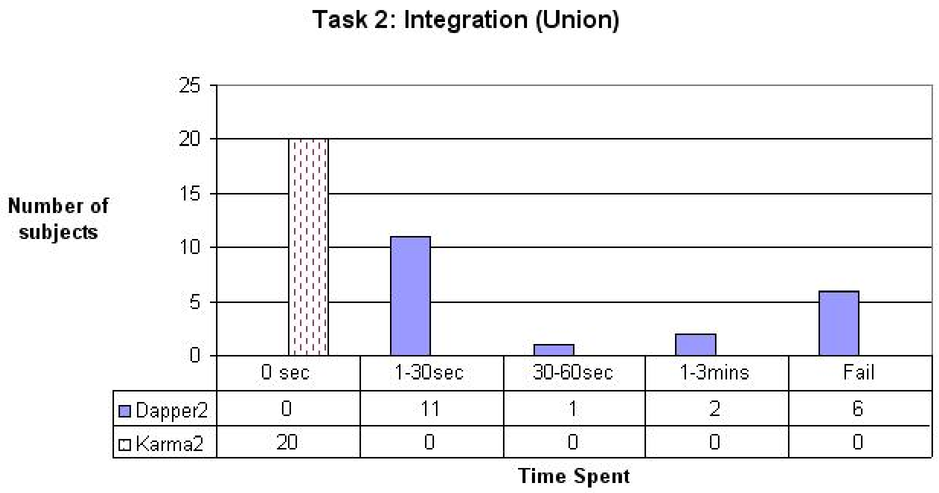
By allowing a subject to enter the cleaned result and having Karma try to infer the cleaning rule, he/she does not have to spend time customizing a cleaning widget. Customizing a cleaning operation in Pipes can be difficult as it requires its users to understand regular expressions; even students who know how to program had difficulty trying to use it.

Karma's cleaning by example does have a limitation. While it is easier to use, it is less expressive than regular expressions; if Karma cannot match the user's example with its predefined rules, then the user would have to manually clean each result. However, for casual Mashup building, providing a predefined set of rules that are most used is a logical trade off compared to spending a long time writing a regular expression to clean the data.

8.6.3.4 Data Integration

Figures 8.21 and 8.22 show the performance comparison between Karma and Dapper/Pipes for the data integration subtask. In each graph, the x-axis is the time spent to complete the data extraction subtask, while the y-axis is the number of subjects that fall into each time segment.

In the database union case shown in Figure 8.21, Karma does not require any time to customize the database union because the spreadsheet structure allows a Karma user to indirectly specify the union by dragging the data from a new similar source into a new row. While it takes more time in Pipes to specify union, the time taken is small compared to the overall time required to build Mashups.



	0 sec	1-30sec	30-60sec	1-3mins	Fail
▨ Dapper2	0	11	1	2	6
□ Karma2	20	0	0	0	0

Figure 8.21: The performance comparison (programmers) between Karma and Dapper/Pipes for the data integration subtask (database union) in Task 2.

Table 8.10 shows the T-test for the hypothesis that Karma is faster than Dapper/Pipes for data integration in each of the task and the overall subtask.

Table 8.10: The statistical significance test result for data integration subtask

Task No	T-test
Task 2 Union (Simple)	t=3.15, degree of freedom=38, and $p < 0.01$
Task 3 Join (Hard)	t=20.51, degree of freedom=38, and $p < 0.01$
Overall data integration	t=7.05, degree of freedom=78, and $p < 0.01$

We also want to show that as the data integration subtask gets progressively harder, Karma performs better in terms of time spent to finish the subtask. Table 8.11 shows the difference between **DiffD** and **DiffK** for the data integration subtask. Using the same approach as the previous subsection, we discard the result from subject No. 3, 10, 12, 18. We found that the increment in time spent using Dapper/Pipes is more than the increment in time spent using Karma for data integration subtasks. In addition, the result is statistically significant (t=4.90, degree of freedom=30, and $p < 0.01$)

In the database join case shown in Figure 8.22, Karma performs much better than Dapper/Pipes; all subjects completed this subtask in less than three minutes, while the majority of the subjects cannot complete the task using Dapper/Pipes. This result shows promise for Karma because one of the main advantages in Mashups is that once built, a Mashup can be reused as a module in another Mashup. However, given the number of existing Mashups, it would be time consuming for a casual user to locate and figure out how to integrate a Mashup built by other people on their own. This is apparent in task 3, where a Dapper/Pipes subject needs to join her own Mashup with another one created by someone else. Karma's approach to data integration allows the subject to bypass the

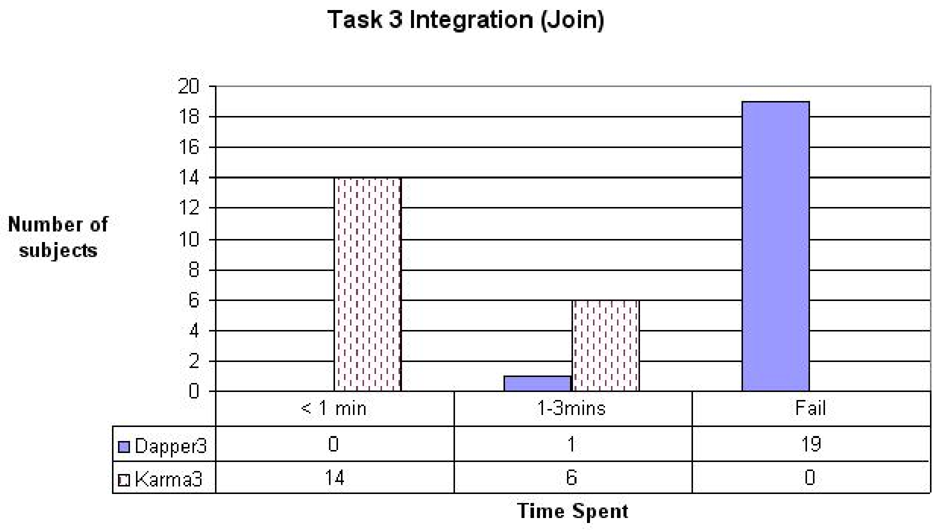


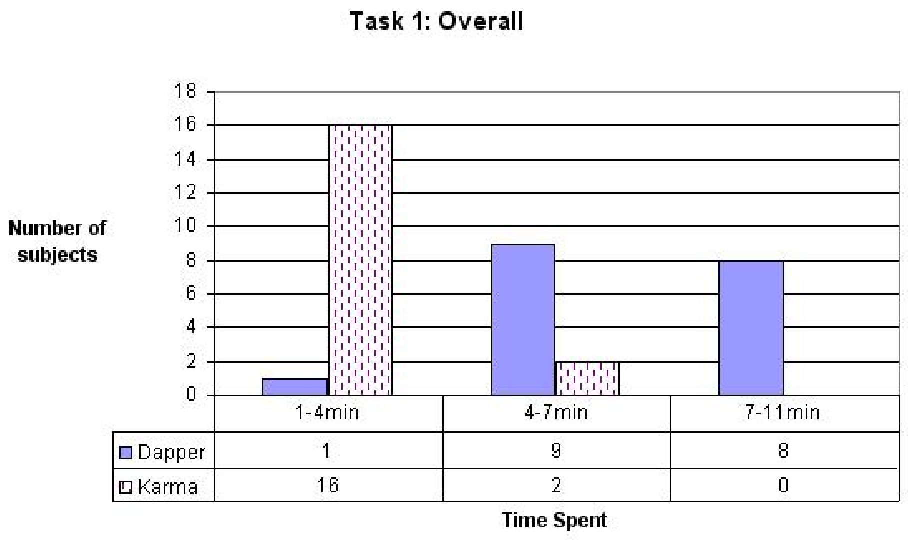
Figure 8.22: The performance comparison (programmers) between Karma and Dapper/Pipes for the data integration subtask (database join) in Task 3.

Table 8.11: The comparision between DiffD and DiffK for each programmer subject for data integration subtasks. The unit is in seconds.

Subject No.	DiffD	DiffK
No.1	225	136
No.2	288	26
~~No.3~~	~~0~~	~~44~~
No.4	262	66
No.5	224	37
No.6	284	31
No.7	14	-6
No.8	282	50
No.9	277	51
~~No.10~~	~~0~~	~~44~~
No.11	280	35
~~No.12~~	~~0~~	~~65~~
No.13	280	46
No.14	290	57
No.15	290	120
No.16	278	45
No.17	273	56
~~No.18~~	~~0~~	~~70~~
No.19	279	80
No.20	278	44
Average	261	57

search step and instead focuses on selecting what kind of data (suggested by Karma) to integrate into a newly extracted data source.

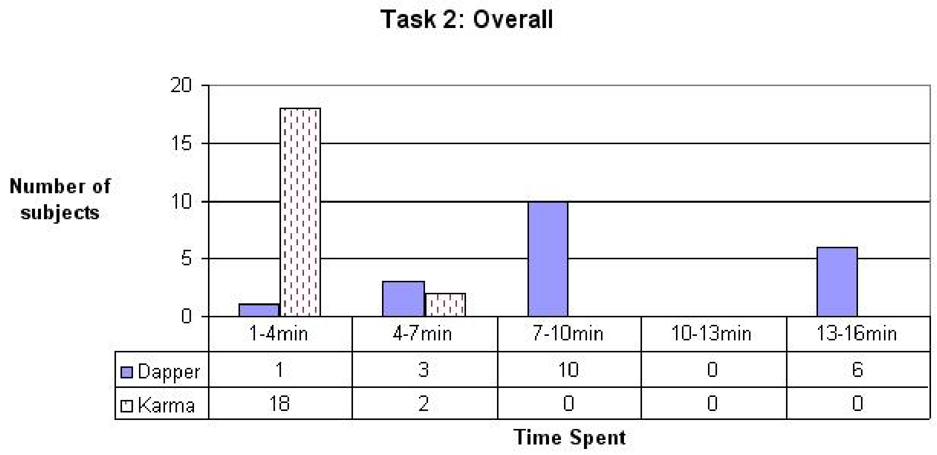
8.6.4 Claim 3: Overall, the user takes less time to build the same Mashup in Karma compared to Dapper/Pipes



	1-4min	4-7min	7-11min
■ Dapper	1	9	8
▨ Karma	16	2	0

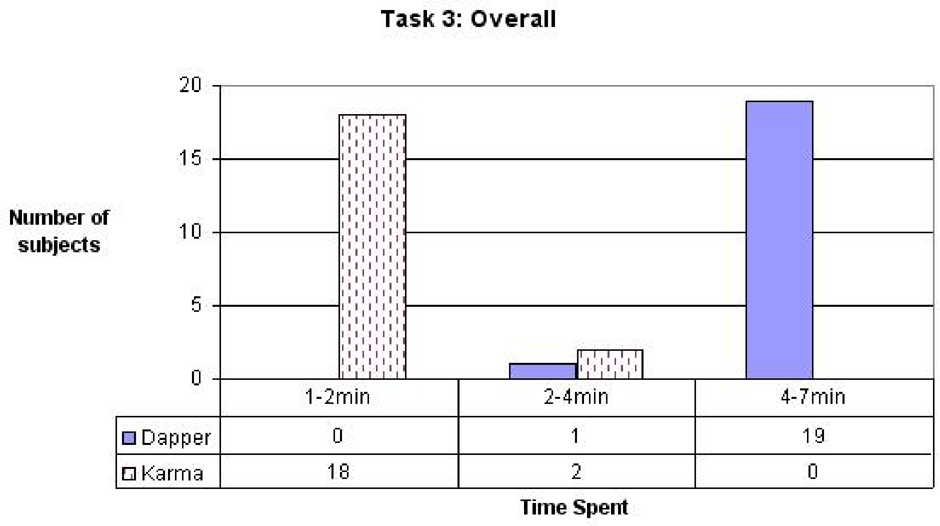
Figure 8.23: The performance comparison (programmers) between Karma and Dapper/Pipes for Task 1.

Figures 8.23, 8.24, and 8.25 show the overall comparison between Karma and Dapper/Pipes in each task. Each graph combines the time spent for each subtask (i.e., data extraction, source modeling, data cleaning, and data integration). For each task, Karma performs noticeably better than Dapper/Pipes; most subjects spent less than 4 minutes to complete each of the tasks using Karma. Table 8.12 shows the T-test for the hypothesis that Karma is faster than Dapper/Pipes for each task and the overall result.



	1-4min	4-7min	7-10min	10-13min	13-16min
■ Dapper	1	3	10	0	6
□ Karma	18	2	0	0	0

Figure 8.24: The performance comparison (programmers) between Karma and Dapper/Pipes for Task 2



	1-2min	2-4min	4-7min
■ Dapper	0	1	19
□ Karma	18	2	0

Figure 8.25: The performance comparison (programmers) between Karma and Dapper/Pipes for Task 3

Table 8.12: The statistical significance test result for each task

Task No	T-test
Task 1	t=9.93, degree of freedom=34, and $p < 0.01$
Task 2	t=7.37, degree of freedom=38, and $p < 0.01$
Task 3	t=23.45, degree of freedom=38, and $p < 0.01$
Overall (combining 3 tasks)	t=13.24, degree of freedom=114, and $p < 0.01$

8.6.5 Failure Rate

Table 8.13 shows the failure rate in Dapper/Pipes. Note that both programmer and nonprogrammer subjects were able to complete all three tasks using Karma without failing. However, subjects who use Dapper/Pipe sometimes fail in a particular subtask. The *Overall* row in Table 8.13 shows the percentage of users who fail on least one subtask in each respective task.

Table 8.13: Individual and Overall failure rate in Dapper/Pipes.

Task	Task 1	Task 2	Task 3
Data Extraction	5.5%(1/18)	25%(5/20)	0%(0/20)
Source Modeling	0%(0/18)	0%(0/20)	0%(0/20)
Data Cleaning	83%(15/18)	35%(7/20)	n/a
Data Integration	n/a	30%(6/20)	95%(19/20)
Overall	83%(15/18)	45%(9/20)	95%(19/20)

Besides the difficulty of customizing widgets, there are three additional factors that contribute to failures in Dapper/Pipes. These factors are described below:

1. Cutoff Failure: In our evaluation, we use the cutoff time of 5 minutes; users who spend more time than 5 minutes in a particular subtask is marked as failing. However, of all 53 instances of failures, there are 11 instances (i.e. 20% of all failure instances) where users were able to complete the subtask using more than 5 minutes. We did not take this extra time into the calculation to ensure a uniform result.

2. Cascading Failure: In task 2, failing to complete the data cleaning subtask can lead to failing to complete the integration subtask. Five out of nine overall failures in task 2 can be attributed to this type of failures. However, task 2 is designed such that users could complete the integration subtask before doing the data cleaning

subtask; users had cascading failures because of their decisions to solve a harder subtask first.

3. Minimal Support Failure: In task 3, we have a failure rate of 95% for Dapper/Pipes. This failure can be attributed to the length it takes users to customize the database join operation. As mentioned in Chapter 1, most Mashup tools choose to focus on particular subtasks while ignoring others. However, the breakdown of Mashup types show that the database join is one of the important features. Karma's support of the database join operation allows users to build Mashups that combine two sources through a join. The other Mashup tool that fully supports database join is Intel's Mashmaker[21]. However, Mashmaker requires experts to customize predefined join operations between sources; casual users cannot customize database join between data sources by themselves.

8.6.6 Summary

Table 8.14 shows a more quantitative comparison between Karma and Dapper/Pipes. The value is computed by averaging the time spent for each subtask over three scenario tasks from results using programmers. The overall result indicates Karma performs better in each of the subtasks (except soure modeling) and in overall tasks. Among the four subtasks, Karma performs extremely well in data cleaning and data integration due to the reasons explained earlier.

While Karma performs worse than Dapper/Pipes for source modeling, the mean difference is only 10 seconds. In addition, this short coming is compensated during the data integration step where Karma took about 2 minutes less on average to complete the

Table 8.14: Overall comparison (programmers) between Dapper/Pipes and Karma average over three tasks. This result does not factor in subjects No.3 and No.17 for Task 1.

Task	Avg time for Dapper/Pipes in minutes	Avg time for Karma in minutes	Factor of improvement
Data Extraction	2:09	0:58	2.22
Source Modeling	0:19	0:28	-0.67
Data Cleaning	4:07	1:00	4.16
Data Integration	3:06	0:29	6.49
Overall	9:41	2:55	3.32

subtask. Overall, it takes our subjects 9:41 minutes on average to build a Mashup using Dapper/Pipes, while it takes our programmer subjects only 2:55 minutes (3:47 minutes for nonprogrammers) to build the same Mashup using Karma. As a result, Karma performs better than Dapper/Pipes by more than a factor of 3. In addition, Karma also allows users to build Mashups that they fail to build using Dapper/Pipes.

Chapter 9

Conclusion and Future Work

To address the four mashup building problem (i.e., data extraction, source modeling, data cleaning, and data integration), I have introduced the Karma framework for building Mashups that focuses on three key ideas. The first idea is to focus on the data, not the operation. By using the programming-by-demonstration paradigm, we can learn the operation that the user wants to perform indirectly by looking at the data provided by the user. The second idea is to leverage existing databases. By comparing the value that the user enters with the data in the existing database, it is possible to deduce some relationships between new data and existing data. These relationships allow Karma to be able to assist users in the problems of source modeling, data cleaning, and data integration. The third idea is to consolidate the Mashup building problems. Since many Mashup building issues are interrelated, it is possible to exploit the structure such that solving a problem in one area can help simplify the process of solving a problem in another area.

I have demonstrated the validity of my approach through a user evaluation that compared my approach with the widget approach using Karma and Dapper/Pipes on two types of users. The experiments have shown that:

- Users can finish three tasks designed to mimic the structure of the four Mashup types using Karma. On the other hand, users have trouble finishing some of the tasks using Dapper/Pipes.
- Users took less time to finish the same task using Karma compared to Dapper/Pipes. The average time used in Karma to build Mashups is 2:55 minutes, while the average time used in Dapper/Pipes is 9:41 minutes.
- Overall, Karma is faster by more than a factor of 3. The key saving areas are data cleaning and data integration. In data cleaning phrase, Karma allows users to specify the result by example and automatically deduces the cleaning operation, while Dapper/Pipes requires users to customize complicated widgets. In data integration phrase, Karma leverages databases to help users decide what data is available to add, while Dapper/Pipes requires users to manually search through existing sources to link with the newly extracted data source.

9.1 Limitations

While my approach makes the Mashup building more effective, it still has some limitations. I have divided limitations into three categories: approach limitation, algorithmic limitation, and coverage limitation.

9.1.1 Approach Limitation

Karma is based on a framework that links each subproblem together using a) programming-by-demonstration and b) a database. This subsection lists the limitations intrinsic to this framework.

- *Dependency on existing databases:* In Karma, its performance in source modeling, data cleaning, and data integration steps partially depend on existing databases as Karma use them to compute suggestions. As a result, if the new data source has no data elements that overlap with existing data sources, users would have to address those problems manually without suggestions from Karma.

- *Fitting concepts into the framework:* Karma uses existing concepts to help solve each subtask. For example, Karma leverages DOM tree to help solve the problem of data extration. There exist other concepts that can potentially be substitued to increase Karma's performance. However, they must be able to adapt to work with a unified table and programming-by-demonstration; it might not be possible to plugin every concepts or algorithms and make them fit the Karma framework.

9.1.2 Algorithmic limitation

Under the unified approach in Karma, Karma uses different algorithms to solve each problem area. This subsection lists the limitation intrinsic to the algorithms used in Karma.

- *Data Extraction:* Karma uses the DOM to help extract data from web sources. Compared to machine learning approach such as Stalker [46], the DOM approach sacrifice some performances to make the interaction more user friendly. For example, if there is a HTML tag irregularity in a particular member of the list, the DOM approach will not be able to extract that member, while [46] could still extract that member by asking it users to label more examples.

- *Source Modeling:* Karma uses simple value comparison to generate candidates for column names. As a result, it cannot deal with the n:m case where the extracted data value is actually a combination of multiple attributes.

- *Data Cleaning:* Karma uses string similarity to help identify potential mispelling. This can be ineffective because it does not take information about domains into account. In addition, cleaning by example relies on a libary of predefined transformations. If Karma cannot induce any transformation, users would have to manually clean the data themselves.

- *Data Integration:* Karma is intended for casual users. As a result, we did not incorporate any concept of quality into the system. In constrast, a query integration system, such as Q [56], allows users to give feedback on the proposed suggestions and uses that feedback to improve futher suggestions.

9.1.3 Coverage limitation

While the focus of this thesis is on the four Mashup types, there are other Mashup building issues common among these four Mashup types that have not been addressed.

- *Dynamic data:* In its current state, users can build Mashups using Karma, but the data is static; the data extracted from web sources does not get refreshed if web sources change. This limitation can be fixed by adding a planning component to Karma so that Karma also captures a re-executable plan to retrieve the new data for each user-built Mashup.

- *Javascript:* Many web sources use Javascript to display data that we might want to extract on their pages. Unfortunately, existing data extraction approaches cannot extract data from Javascript.

9.2 Future Work

9.2.1 Customize Display by Examples

In this thesis, we focus on generating a table of data which is the backbone of a Mashup. While Karma provides a means to display the data in the table on the Map, users cannot fine tune how the display should look like and how the display should behave. We believe that we can apply the ideas used in Karma framework to address the data display problem too. For example, programming-by-demonstration can be used to help Karma learn from examples the kind of display that users might want to specify. In addition, a database that contains past choices made by users can be used to match the pattern that users are performing during the customization and suggest display operations.

9.2.2 Recovering From Errors

One of the limitations in Karma is recovering from errors. Karma uses heuristics to capture Mashup building operations, but these heuristics can be wrong. Recovering from errors is an active area of research in the programming-by-demonstration domain.

However, in existing systems, users often need to trace through a concept tree induced by PBD's heuristic to fix these errors. As a result, it will be difficult for casual users to trace and recover from errors using existing frameworks. A new interaction framework that allows casual users to browse through different result scenarios instead of searching through an error tree might be a possible approach to solve this problem.

9.2.3 Feedbacks and Quality

Not all data sources are created equal. Some data sources might be useful or pertinent to users in one task, but not in another task. When integrating data from multiple sources, Karma treats each data sources as equal. A concept of feedback and the quality of sources can be incorporated in Karma to make the data integration step more robust and more accurate.

References

[1] http://www.wikipedia.org.

[2] Serge Abiteboul, Sophie Cluet, Tova Milo, Pini Mogilevsky, Jme Simeon, and Sagit Zohar. Tools for data translation and integration. IEEE Data Engineering Bulletin, 22(1):3–8, 1999.

[3] Eugene Agichtein and Venkatesh Ganti. Mining reference tables for automatic text segmentation. In KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining, pages 20–29. ACM, 2004.

[4] James Allen, Nathanael Chambers, George Ferguson, Lucian Galescu, Hyuckchul Jung, Mary Swift, and William Tayson. Plow: A collaborative task learning agent. In Proceedings of Conference on Artificial Intelligence (AAAI), pages 22–26. Springer-Verlag, 2007.

[5] Rahul Bakshi, Craig A. Knoblock, and Snehal Thakkar. Exploiting online sources to accurately geocode addresses. In GIS '04: Proceedings of the 12th annual ACM international workshop on Geographic information systems, pages 194–203. ACM, 2004.

[6] F. Benzi, D. Maio, and S. Rizi. Vistool: a visual tool for querying relational databases. In Proceedings of Advanced Visual Interface, pages 258–260. ACM Press, 1998.

[7] Sonia Bergamaschi, Silvana Castano, Maurizio Vincini, and Domenico Beneventano. Semantic integration of heterogeneous information sources. Data & Knowledge Engineering, 36(3):215–249, 2001.

[8] Gaurav Bhalotia, Arvind Hulgeri, Charuta Nakhe, Soumen Chakrabarti, S. Sudarshan, and I.I.T. Bombay. Keyword searching and browsing in databases using banks. icde, 00:0431, 2002.

[9] Mikhail Bilenko and Raymond J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, pages 39–48. ACM, 2003.

[10] M. Burnett and M. Baker. A classification system for visual programming languages. Technical report, Oregon State University, 1993.

[11] Mark James Carman and Craig A. Knoblock. Learning semantic definitions of online information sources. Journal of Artificial Intelligence Research (JAIR), 2007.

[12] Surajit Chaudhuri and Umeshwar Dayal. An overview of data warehousing and olap technology. SIGMOD Record, 26(1):65–74, 1997.

[13] Jiun-Hung Chen and Daniel S. Weld. Recovering from errors during programming by demonstration. In IUI '08: Proceedings of the 13th international conference on Intelligent user interfaces, pages 159–168. ACM, 2008.

[14] W. Chen, M. Kifer, and D. S. Warren. Hilog: A foundation for higher-order logic programming. Journal of Logic Programming, pages 187–230, 1993.

[15] W. Cohen, P. Ravikumar, and S. Fienberg. A comparison of string distance metrics for name-matching tasks. In Proceedings of the International Joint Conferences on Artificial Intelligence (IJCAI), pages 73–78. American Association for Artificial Intelligence, 2003.

[16] V. Crescenzi, G. Mecca, , and P. Merialdo. Roadrunner: Towards automatic data extraction from large web sites. Technical report, Universit a di Roma Tre, 2001.

[17] A. Cypher. Watch what I do: Programming by demonstration. MIT Press, 1993.

[18] Robin Dhamankar, Yoonkyong Lee, AnHai Doan, Alon Halevy, and Pedro Domingos. imap: discovering complex semantic matches between database schemas. In SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data, pages 383–394. ACM, 2004.

[19] A. Doan, P. Domingos, and A.Y. Levy. Learning source descriptions for data integration. In Proceedings of the International Workshop on The Web and Databases (WebDB), pages 81–86. AAAI Press, 2000.

[20] Mira Dontcheva, Steven M. Drucker, David Salesin, and Michael F. Cohen. Relations, cards, and search templates: user-guided web data integration and layout. In UIST '07: Proceedings of the 20th annual ACM symposium on User interface software and technology, pages 61–70. ACM, 2007.

[21] Rob Ennals and David Gay. User-friendly functional programming for web mashups. In ICFP '07: Proceedings of the 2007 ACM SIGPLAN international conference on Functional programming, pages 223–234. ACM, 2007.

[22] Oren Etzioni and Ruth Etzioni. Statistical methods for analyzing speedup learning experiments. Mach. Learn., 14(3):333–347, 1994.

[23] G. Fischer and H. Nieper-Lemke. Helgon: extending the retrieval by reformulation paradigm. In CHI '89: Proceedings of the SIGCHI conference on Human factors in computing systems, pages 357–362. ACM, 1989.

[24] Andrew Gibson, Matthew Gamble, Katy Wolstencroft, Tom Oinn, and Carole Goble. The data playground: An intuitive workflow specification environment. In E-SCIENCE '07: Proceedings of the Third IEEE International Conference on e-Science and Grid Computing, pages 59–68. IEEE Computer Society, 2007.

[25] Alon Halevy, Anand Rajaraman, and Joann Ordille. Querying heterogeneous information sources using source descriptions. In Proceedings of the International Conference on Very Large Database (VLDB), pages 251–262. Morgan Kaufmann, 1996.

[26] Alon Halevy, Anand Rajaraman, and Joann Ordille. Data integration: The teenage years. In Proceedings of the International Conference on Very Large Database (VLDB), pages 9–16. Morgan Kaufmann, 2006.

[27] Björn Hartmann, Leslie Wu, Kevin Collins, and Scott R. Klemmer. Programming by a sample: rapidly creating web applications with d.mix. In UIST '07: Proceedings of the 20th annual ACM symposium on User interface software and technology, pages 241–250. ACM, 2007.

[28] David Haw, Carole A. Goble, and Alan L. Rector. Guidance: Making it easy for the user to be an expert. In Proceedings of International Workshop on Interfaces to Database, pages 25–48, 1994.

[29] A. He and N. Kushmerick. Learning to attach semantic metadata to web services. In Proceedings of 2nd International Semantic Web Conference (ISWC). Springer-Verlag, 2003.

[30] David Huynh, Stefano Mazzocchi, and David Karger. Piggy bank: Experience the semantic web inside your web browser. Web Semantics, 5(1):16–27, 2007.

[31] David F. Huynh, Robert C. Miller, and David R. Karger. Potluck: Data mash-up tool for casual users. In Proceedings of the International Semantic Web Conference. Springer-Verlag, 2007.

[32] Jihie Kim and Marcelo Tallis. User studies of knowledge acquisition tools: Methodology and lessons learned. Experimental and Theoretical Artificial Intelligence (JETAI), 2001.

[33] V.S. Lakshmanan, F. Safris, and I.N. Subramaniant. Schemasql: A language for intereoperability in relational multi-database systems. In VLDB '96: Proceedings of the 22th International Conference on Very Large Data Bases, pages 239–250. Morgan Kaufmann, 1996.

[34] T. Lau, L. Bergman, V Castelli, and D. Oblinger. Programming shell scripts by demonstration, workshop on supervisory control of learning and adaptive systems. In Proceedings of Conference on Artificial Intelligence (AAAI). AAAI Press, 2004.

[35] Tessa Lau. Programming by Demonstration: a Machine Learning Approach. PhD thesis, University of Washington, 2001.

[36] Tessa Lau, Lawrence Bergman, Vittorio Castelli, and Daniel Oblinger. Sheepdog: learning procedures for technical support. In IUI '04: Proceedings of the 9th international conference on Intelligent user interfaces, pages 109–116. ACM, 2004.

[37] Kristina Lerman, Craig A. Knoblock, and Steve Minton. Automatic data extraction from lists and tables in web sources. In Proceedings of IJCAL Workshop on Adaptive Text Extraction and Mining. AAAI Press, 2001.

[38] W. Li, C. Clifton, and S. Liu. Database integration using neural network: Implementation and experiences. Knowledge and Information Systems.

[39] Henry Lieberman. Your Wish is My Command: Programming By Example. Morgan Kaufman, 2001.

[40] Andrew McCallum, Kedar Bellare, and Fernando C. N. Pereira. A conditional random field for discriminatively-trained finite-state string edit distance. In Conference on Uncertainty in Artificial Intelligence (UAI), pages 388–395. AUAI Press, 2005.

[41] Martin Michalowski, Craig A. Knoblock, Ken Bayer, and Berthe Y. Choueiry. Exploiting automatically inferred constraint-models for building identification in satellite imagery. In GIS '07: Proceedings of the 15th annual ACM international symposium on Advances in geographic information systems, pages 1–8. ACM, 2007.

[42] Martin Michalowski, Snehal Thakkar, and Craig A. Knoblock. Exploiting secondary sources for unsupervised record linkage. In Proceedings of the 2004 VLDB Workshop on Information Integration on the Web, pages 34–39, 2004.

[43] Matthew Michelson and Craig A. Knoblock. Unsupervised information extraction from unstructured, ungrammatical data sources on the world wide web. International Journal of Document Analysis and Recognition (IJDAR), Special Issue on Noisy Text Analytics, 2007.

[44] Matthew Michelson and Craig A. Knoblock. Creating relational data from unstructured and ungrammatical data sources. Journal of Artificial Intelligence Research (JAIR), 31:543–590, 2008.

[45] T. Milo and S. Zohar. Using schema matching to simplify heterogeneous data translation. In Proceedings of the International Conference on Very Large Database (VLDB), pages 122–133. Morgan Kaufmann, 1998.

[46] Ion Muslea, Steven Minton, and Craig A. Knoblock. Active learning with strong and weak views: A case study on wrapper induction. In Proceedings of the 18th International Joint Conference on Artificial Intelligence, pages 415–420. Lawrence Erlbaum Associates LTD, 2003.

[47] Jeffrey Nichols and Tessa Lau. Mobilization by demonstration: using traces to re-author existing web sites. In IUI '08: Proceedings of the 13th international conference on Intelligent user interfaces, pages 149–158. ACM, 2008.

[48] Daniel E. O'Leary. The use of mathematical programming to verify rule-based knowledge. Annals of Operations Research, 1996.

[49] A. Papantonakis and P.J.H. King. Syntax and semantics of gql: a graphical query language. Journal of Visual Languages, 1995.

[50] J. Perkowitz and O. Etzioni. Category translation: Learning to understand information on the internet. In Proceedings of International Joint Conferences on Artificial Intelligence (IJCAI), pages 930–936. Morgan Kaufmann, 2005.

[51] Erhard Rahm and Phillip A. Bernstein. On matching schemas automatically. VLDB Journal.

[52] Vijayshankar Raman and Joseph M. Hellerstein. Potter's wheel: An interactive data cleaning system. In VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases, pages 381–390. Morgan Kaufmann Publishers Inc., 2001.

[53] Alberto Segre, Charles Elkan, and Alexander Russell. A critical look at experimental evaluations of ebl. Machine Learning, 6(2):183–195, 1991.

[54] Atsushi Sugiura and Yoshiyuki Koseki. Internet scrapbook: automating web browsing tasks by demonstration. In UIST '98: Proceedings of the 11th annual ACM symposium on User interface software and technology, pages 9–18. ACM, 1998.

[55] William Robert Sutherland. The On-Line Graphical Specification of Computer Procedures. PhD thesis, Massachusetts Institute of Technology, 1966.

[56] Partha P. Talukdar, Marie Jacob, Mohammad S. Mehmood, Koby Crammer, Zachry G. Ives, Fernando Pereira, and Sudipto Guha. Learning to create data-integrating queries. In Proceedings of the VLDB., 2008.

[57] Snehal Thakkar, Jose Luis Ambite, and Craig A. Knoblock. Composing, optimizing, and executing plans for bioinformatics web services. The International Journal on Very Large Data Bases (VLDB).

[58] Rattapoom Tuchinda and Craig A. Knoblock. Agent wizard: building information agents by answering questions. In IUI '04: Proceedings of the 9th international conference on Intelligent user interfaces, pages 340–342. ACM, 2004.

[59] J.D. Ullman. Principles of Database and Knowledge-Base Systems Volume I. Computer Science Press, 1988.

[60] Michael D. Williams and Frederich N. Tou. Rabbit: An interface for database access. In ACM 82: Proceedings of the ACM '82 conference, pages 83–87, New York, NY, USA, 1982. ACM.

[61] W. Winkler. Overview of record linkage and current research directions. Technical report, Statistical Research Division, U.S. Bureau of the Census, 2006.

[62] Jeffrey Wong and Jason I. Hong. Making mashups with marmite: towards end-user programming for the web. In CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems, pages 1435–1444. ACM, 2007.

[63] R.F. Woolson and P.A. Lachencruch. Rank tests for censored matched pairs. Biometrika, 1980.

[64] L. Xu and D. Embley. Using domain ontologies to discover direct and indirect matches for schema elements. In Proceedings of the Semantic Integration Workshop at ISWC, 2003.

[65] Ling Ling Yan, Renée J. Miller, Laura M. Haas, and Ronald Fagin. Data-driven understanding and refinement of schema mappings. In SIGMOD '01: Proceedings of the 2001 ACM SIGMOD international conference on Management of data, pages 485–496. ACM, 2001.

[66] Moshé M. Zloof. Query-by-example: the invocation and definition of tables and forms. In VLDB '75: Proceedings of the 1st International Conference on Very Large Data Bases, pages 1–24. ACM, 1975.