RC24693 (W0811-074) November 11, 2008
Computer Science

IBM Research Report

An Interactive, Smart Notepad for Context-Sensitive Information Seeking

Jie Lu
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598
USA

Michelle X. Zhou
IBM China Research Lab
Shandi Zhong Guan Cun Software Park
Hsidian District
Beijing 100192
P.R. China

Research Division
Almaden - Austin - Beijing - Cambridge - Haifa - India - T. J. Watson - Tokyo - Zurich

An Interactive, Smart Notepad for Context-Sensitive Information Seeking

Jie Lu
IBM T. J. Watson Research Center
19 Skyline Drive
Hawthorne, NY 10532
jielu@us.ibm.com

Michelle X. Zhou
IBM China Research Lab
Shandi Zhong Guan Cun Software Park
Haidian District, Beijing 100192
mxzhou@cn.ibm.com

ABSTRACT

We are building an interactive, smart notepad system where users enter brief notes to drive a dynamic information-seeking process. In this paper, we focus on describing our work from two aspects: 1) dynamic interpretation of user notes in context to infer a user's information needs, and 2) automatic generation of data queries to satisfy the inferred user needs. Compared to existing information systems, our work offers three unique contributions. First, our system allows users to focus on *what* to retrieve instead of *how*, since users can use brief notes to express their information needs without worrying about specific retrieval details. Second, users can use notes to efficiently request multiple pieces of information *at once* instead of issuing one query at a time. Third, users can easily update any part of their notes to obtain new or updated information. Whenever a user's notes are modified, our system automatically detects and evaluates all affected sections including the existing notes to retrieve new or updated information. Our preliminary evaluation shows the promise of our work.

1. INTRODUCTION

Information seeking is a routine task in many lines of work. However, information seeking can often be time consuming and difficult for several reasons.

First, an information-seeking task (e.g., house hunting) may require the retrieval of multiple pieces of information about different but related data entities and aspects. However, today's information systems are usually designed to handle one query at a time. A user thus must manually craft and issue multiple queries, one at a time, and then manually stitch together information obtained at different steps.

Second, a user's information needs may change during an information-seeking process. To meet the evolving information needs, the user may revisit previously issued queries and modify them to obtain new or updated information. Nonetheless, few existing systems allow users to easily revise their previous inquiries in context. Without such support, users may have to start over again.

Third, a user's new/updated request (e.g., a flight request) may change the context of previous requests (e.g., hotel request). Accordingly, these previously satisfied requests may need to be re-evaluated in the light of the new context. Again, few existing systems *automatically* track the context of a request, let alone re-evaluating it when the context changes. As a result, users themselves must manually track their requests and update them when needed.

```
1  Restaurants near IBM Hawthorne
2
3  Italian
4  name, address, phone number, avg customer rating
5  Tramonto, 27 Saw Mill River Rd., (914)347-8220, 3.5
6
7  Chinese
8  Oriental Diner, 58 Saw Mill River Rd., (914)769-0038, 3.0
```

Figure 1. User notes (in black) entered to gather restaurant information (in blue). Texts in bold face are the user input and system output at the current turn.

To address the above challenges and better aid users in their information-seeking tasks, we are building a smart notepad system called CENTAUR (*C*ontent *EN*semble *T*ool for *A*daptive *U*ser-centered *R*etrieval). Using an interactive notepad as the interface metaphor, CENTAUR lets users interactively express their information requests in a form of notes (Figure 1 lines 1–4, 7). It then dynamically interprets the user notes to retrieve the requested information and presents it in the notepad (Figure 1 lines 5 and 8).

Compared to existing systems, CENTAUR offers users three distinctive benefits. First, it allows users to easily specify their explicit or implicit information needs in brief notes without worrying about the underlying retrieval details. For example, line 7 in Figure 1 is used to request the information about chinese restaurants near IBM Hawthorne. Second, users can efficiently collect multiple pieces of information at once. For example, line 4 is used to gather multiple pieces of restaurant data. Third, users can easily modify any part of their notes in context to obtain new/updated information. Consider the new input "*avg customer rating*" at line 4. This update retrieves the rating information for both italian and chinese restaurants (lines 5 and 8). In such cases, CENTAUR automatically identifies and evaluates affected note sections (e.g., line 4 and below) to retrieve the newly requested information (e.g., ratings).

To support the functions of a smart notepad as described above, CENTAUR faces two main challenges. First, a user's notes are usually given in a form of informal text (Figure 1). It is a challenge for CENTAUR to accurately interpret the

meanings of such input. Moreover, a user's input may lump together multiple requests (e.g., line 4), which must be identified and fulfilled by one or more data queries. This process becomes more challenging, if the needed information is stored in different formats or to be gathered from multiple data sources (e.g., relational databases vs. text collections).

Second, a user's notes are often context sensitive. This requires CENTAUR to interpret the notes in context to derive their full meanings. For example, the input "*Chinese*" at line 7 in Figure 1 implies the need of information for chinese restaurants near IBM Hawthorne. To retrieve the requested information, CENTAUR must use other parts of the notes (lines 1–4) to formulate the correct queries.

In this paper, we describe how CENTAUR tackles these challenges from two aspects:

- **Dynamic, context-sensitive user note interpretation.** CENTAUR uses a semantics-based approach to model and interpret user notes in context. As a result, it derives a set of user information requests.
- **Automatic query generation.** Based on the derived information requests, CENTAUR uses a hybrid of rule-based and procedural approach to automatically generate executable queries. Currently, it can generate two types of queries: SQL and keyword-based queries.

We have applied CENTAUR to two domains. One is a healthcare application where hospital physicians jot down notes to interactively gather pertinent patient data for composing daily patient summaries. The other is a hospitality application where users input notes to interactively collect relevant hotel and restaurant information. For the sake of comprehension, in this paper we use examples from the hospitality domain to illustrate our work.

In the rest of the paper, we first briefly discuss the related work, followed by an overview of CENTAUR. We use a set of concrete examples to illustrate how CENTAUR aids users in their information-seeking tasks. We then describe CENTAUR's key features and our evaluations.

2. RELATED WORK

Our work is closely related to a number of systems that support exploratory information search (e.g., [2, 15, 23]). Similar to these systems, CENTAUR provides a unified environment for users to interactively manage their context, including their queries and retrieval results. Unlike these systems, which are driven by explicit user queries, CENTAUR *automatically* infers a user's information needs and generates corresponding queries based on the user's notes. In addition, previous systems normally handle one query at a time. In contrast, CENTAUR can generate and execute *multiple* queries at once based on a user's input.

Since CENTAUR allows users to express their requests in natural language fragments, it is related to information systems with an natural language interface. These systems can be classified into two types. The first type of systems handles only independent natural language queries (e.g., [14, 22]). The second type of systems processes a user's natural language queries in context (e.g., [1, 11, 13]).

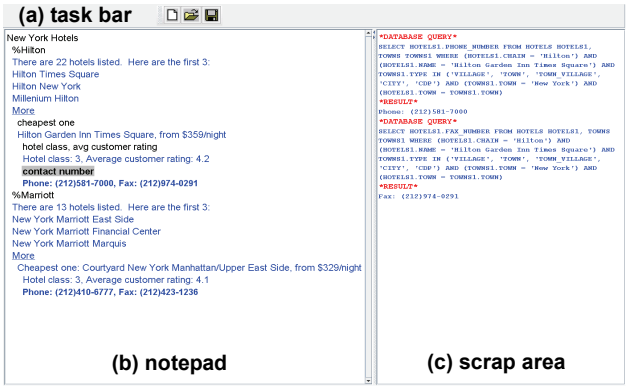


Figure 2. CENTAUR user interface.

CENTAUR is an extension of the second type of systems. It extends these systems in two key aspects. First, these systems are driven by explicit user queries, one query at a time. In contrast, CENTAUR allows users to specify multiple information requests together. Second, these systems do not re-evaluate past queries in an updated context (e.g., a user's new input). In comparison, CENTAUR dynamically assesses whether a user's new input affects the interpretation of the previously entered requests. It then adaptively reinterprets the affected requests in the new context to retrieve new/updated information.

Other related efforts include context-sensitive information retrieval systems and content recommendation systems that exploit various contextual information to provide users with more relevant information. For example, there are information retrieval systems designed to exploit a user's interest profile [7], query history [19, 20, 21], search activities [5, 8, 9, 17, 21], and interactions [12, 18, 23]. Content recommendation systems automatically generate queries based on the information that a user is reading or composing (e.g., web browsing [3, 4, 16], emailing [6, 16], and paper writing [4]). Compared to these systems, CENTAUR uses a much finer-grained context based on the content and structure of a user's notes. As a result, it can infer a user's information needs more accurately.

3. CENTAUR OVERVIEW

Here we provide an overview of CENTAUR, starting with its user interface, followed by its system architecture.

3.1 User Interface

As shown in Figure 2, CENTAUR provides users with three interaction areas: the task bar (Figure 2a), the notepad (Figure 2b), and the scrap area (Figure 2c).

The *task bar* (Figure 2a) hosts a set of controls. The controls allow a user to create a new note file, load notes from a file, or save the current notes into a file.

The *notepad* (Figure 2b) is the main interaction area where users interactively input/modify their notes (black text), and view retrieved information (blue text). It supports main editing operations of a typical text editor, including text insertion, deletion, copying, and pasting. Users can enter notes in the form of text fragments, including natural language

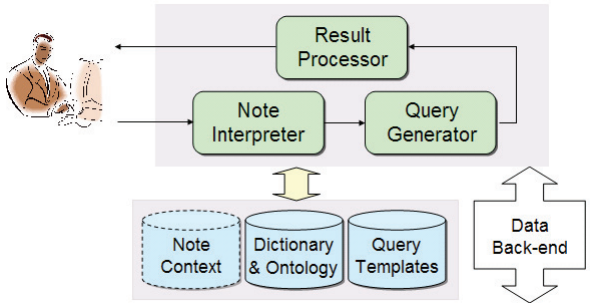


Figure 3. CENTAUR architecture.

phrases, keywords, and lists. At any time during his/her note composition, a user can press the Esc key to ask CENTAUR to retrieve the needed information based on the notes entered so far. Currently, information is retrieved only when a user explicitly requests it. This gives users the full control of deciding when the system's help is needed without being arbitrarily interrupted during their note composition.

Depending on the retrieval results, CENTAUR presents them differently. If a single data item is retrieved (e.g., the cheapest hotel), CENTAUR inserts the item in the notepad right below the corresponding user-entered notes. If multiple data items are retrieved (e.g., a list of hotels), CENTAUR inserts the total count and first three items in the notepad, and provides a hyperlink to the full result list.

For debugging purpose, a *scrap area* (Figure 2c) displays all queries (e.g., SQL queries) generated by CENTAUR and the corresponding query results. A user can highlight a note fragment in the notepad to view CENTAUR-generated queries and the query results. The user can also directly modify the generated queries to obtain new results.

3.2 Architecture

Figure 3 provides an overview of CENTAUR architecture. It has three main modules: note interpreter, query generator, and result processor.

A user starts by entering his/her notes in the notepad. At any time, the user can press the Esc key to ask CENTAUR to retrieve the needed information based on the entered notes. Accordingly, the *note interpreter* processes the user input and derives its meaning. In CENTAUR, the meaning of a user's input encodes the user's information needs to be fulfilled. Based on the derived information needs, the *query generator* builds system-executable queries. After the queries are executed, the retrieval results are passed to the *result processor* for post processing (e.g., selecting the top-*N* items from the result list). The processed results are then displayed to the user.

CENTAUR maintains a set of knowledge sources to help interpret user notes and generate queries. They include a note context model that is dynamically built and updated during a user's information-seeking process, a domain-specific data ontology and a dictionary for understanding user notes, and a set of domain-independent query templates for generating system-executable queries. CENTAUR is also connected to an application data back-end via a database server (e.g., IBM DB2) or a search engine (e.g., Lucene).

4. EXAMPLE SCENARIOS

In this section, we use a set of concrete examples in the hospitality domain to illustrate how CENTAUR aids users in their information-seeking tasks. A running version of these examples can be seen in our submitted video at http://boston.lti.cs.cmu.edu/jielu/iui09.html.

Interpreting User Notes in Context

CENTAUR lets users specify their information needs in text notes, including keywords, natural language phrases, and lists. Consider Alison who is exploring hotels and restaurants for her upcoming vacation in the city of New York. She starts by entering the city name and desired hotel chain (Figure 4a). She then presses the Esc key to ask CENTAUR to retrieve the needed information.

Upon Alison's request, CENTAUR processes her input and automatically generates queries to retrieve the requested information. In this case, CENTAUR formulates a SQL query to retrieve Hilton hotels in New York. Consequently, twenty-two hotels are found. Since this request brings back multiple data items, CENTAUR displays the hotel count, first three hotels, and a hyperlink ("*More*") to the full hotel listings (Figure 4a).

In Alison's input, shorthand notations like "*%*" explicitly indicate desired information types. For example, "*%*" in "*%Hilton*" denotes a data constraint (*chain*="*Hilton*"). Although not required, the use of shorthand notations helps reduce input ambiguities, which in turn improves CENTAUR's note interpretation accuracy.

Similar to entering notes on paper, users may input new notes in the context of existing ones. Assume that Alison now wants to see the cheapest Hilton hotel in New York. She enters "*cheapest one*" and then presses Esc (Figure 4b).

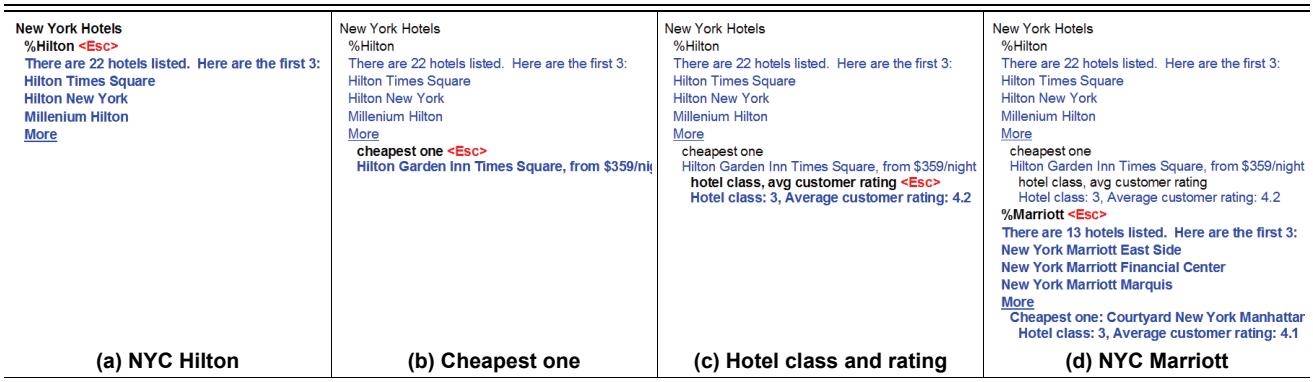


Figure 4. Screenshots of a user's evolving note input (in black) and CENTAUR-retrieved information (in blue).

This input by itself is incomplete, and meant to be interpreted in the context of the previously entered notes.

By default, CENTAUR always interprets a piece of notes in the context of *existing* notes. However, users can explicitly indicate the applicable context when entering a note. For example, a user can use indentations to indicate the indented lines (e.g., "*cheapest one*" in Figure 4b) to be interpreted in the context of all its ancestors (e.g., user input in Figure 4a).

Identifying Multiple Information Requests

Instead of issuing one query at a time, a user may use notes to express multiple information requests at once. Suppose that Alison wants to know the hotel class and average customer rating of the cheapest Hilton hotel in New York. She lumps together the two requests in "*hotel class, avg customer rating*" (Figure 4c). Accordingly, CENTAUR builds two queries to retrieve the hotel class information and compute the average customer rating, respectively.

Besides specifying multiple information requests explicitly, a user may also imply multiple requests in his/her notes. Suppose that Alison wants the same set of information on Marriott as on Hilton. Although implicit in her input (Figure 4d), CENTAUR understands her implied requests and retrieves four pieces of information on Marriott all together (Figure 4d): (1) hotel count and listings, (2) cheapest one, (3) hotel class, and (4) average customer rating.

Handling Note Update

At any given time, a user can edit any part of the existing notes to indicate the change of his/her information needs. Accordingly, CENTAUR retrieves new information based on the updated notes. Assume that Alison decides to visit San Francisco instead. She would then replace "*New York*" with "*San Francisco*" at line 1 in Figure 4(d). Based on this update, CENTAUR would re-process the entire notes entered so far and retrieve the relevant information on Hilton and Marriott hotels in San Francisco.

The above example shows that Alison can easily modify her existing notes to satisfy her updated information needs. She can also insert new input *in the middle of* the existing notes. Suppose that Alison wants to get the contact information for the cheapest hotels. To do so, she inserts the text "*contact number*" under the Hilton block (Figure 5). Given this input, CENTAUR retrieves the requested information for the cheapest Hilton hotel. It also automatically re-interprets all the notes entered *below* the insertion point. The rationale is that the new input may have altered the context of the notes below. In this case, CENTAUR re-interprets "*%Marriott*" in the context of the new input. As a result, it also retrieves the contact number for the cheapest Marriott hotel (Figure 5). Here, the phrase "*contact number*" is ambiguous, since it may refer to the phone number or the fax number. CENTAUR thus retrieves both numbers.

5. CONTEXT-SENSITIVE NOTE INTERPRETATION

As described above, a user can use notes to express his/her information needs explicitly and implicitly. To identify and satisfy user information needs, CENTAUR dynamically interprets the user's notes in context and then generates corresponding queries to retrieve the desired information. In this section, we describe how CENTAUR interprets a user's notes to infer his/her information needs. We explain our approach in two steps. First, we introduce a graph-based representation for modeling a user's information needs. Second, we show how CENTAUR dynamically interprets a user's notes to build such a graph-based structure.

Figure 5. Handling new input "*contact number*".

5.1 Representing A User's Information Needs

To facilitate note interpretation and query generation, we model a user's information needs from two aspects. First, we define an *information request* to denote a user's information need that can usually be fulfilled by a *single* query. Second, we model a *note context* that is an aggregation of *all* information requests derived from a user's note input.

Representing An Information Request

Precisely, an information request encodes one of the following: (1) a data set of the *same* concept with/without constraints (e.g., a hotel set), (2) a single data attribute of a data set[1] defined by (1) (e.g., hotel class of a hotel set), or (3) text information relevant to a single data concept with/without constraints (e.g., reviews of hotels). While information for types 1 and 2 can be obtained by database queries, type 3 information is usually retrieved by keyword-based queries.

We use a graph-based model to represent an information request. Figure 6(a) shows an example of such representation. The graph consists of a set of nodes and links. Each node represents a data concept (e.g., *Hotel*), and each link encodes the semantic relationship between two concepts (e.g., *Hotel located-in City*). In addition, a node may contain a set of data attributes (e.g., *name*) and data constraints (e.g., *chain="Hilton"*). A data attribute could be an aggregated attribute, which includes an aggregation operator and a data attribute, like *min*(*roomRate*). A data constraint consists of an operator (e.g., *EQ*) and a set of operands. An operand can be a data concept, a data attribute, or a constant value.

Each information request has an *anchor node* that defines the type of information to be retrieved. For example, the *Hotel* is the anchor node in Figure 6(a), requesting the relevant hotels to be retrieved. The retrieval results of an information request are also stored in the request. Furthermore,

1 Theoretically, multiple data attributes of the same data set could be retrieved using a single query. However, when aggregated attributes (e.g., min and max room rates) are involved, it becomes complex. For practical reasons, we now limit the retrieval to one attribute per query.

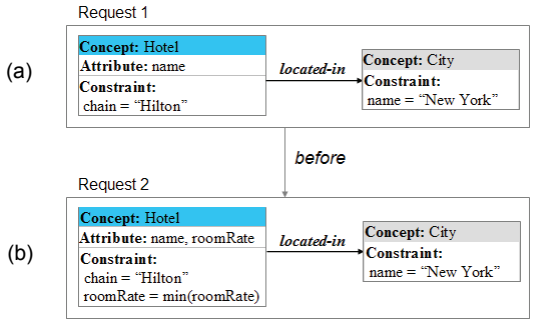


Figure 6. Note context built based on the note input in Figure 4(a–b). The temporal relationship "*before*" indicates that Request 1 (Figure 4a) occurred before Request 2 (Figure 4b).

each information request is associated with a set of features. Currently CENTAUR uses three features: input-location, indentation-level, and data-source (Table 1). Features *input-location* and *indentation-level* are used to determine the relationship between two information requests (Sec. 5.2). Feature *data-source* helps generate a data query (Sec. 6). The values of all three features are dynamically derived during CENTAUR's note processing or query generation stage.

Representing the Context of User Notes

As described in Section 4, a user's notes may specify multiple information requests. Moreover, a user may enter notes in the context of existing notes. We thus use an aggregated graph to represent *all* information requests derived from a user's notes entered so far. Figure 6 is an aggregated graph that contains two sub-graphs, representing two derived information requests: 1) *Hilton hotels in New York* (Figure 6a), and 2) *the cheapest Hilton hotel in New York* (Figure 6b). Here, the first request (Request 1) is derived earlier based on Alison's input shown in Figure 4(a). Using Request 1 as the context, CENTAUR derives Request 2 given her new input in Figure 4(b). The two requests are linked together by their temporal relationship. This aggregated graph will then serve as the context to interpret new notes (e.g., input in Figure 4c).

5.2 Deriving Information Requests in Context

Based on the representation described above, CENTAUR interprets a user's notes to derive one or more information requests. Currently, we extend our own natural language query interpretation engine [13] to note interpretation. Our extensions are made in three areas to accommodate the unique characteristics of user note input.

First, we augment the original approach, which handles only one user request at a time, to derive *multiple* information requests specified by a user's notes. Second, we improve the existing context integration algorithm [10] to exploit unique characteristics of note input, such as the note structure. CENTAUR thus can accurately interpret a user's notes in context. Third, we extend our original work to automatically re-interpret existing notes based on new user input. As a result, CENTAUR can adaptively retrieve relevant information to meet a user's evolving information needs.

Feature	Value
input-location	location of the notes which the request is derived from
indentation-level	indentation level of the notes which the request is derived from
data-source	data sources that may be used to satisfy the request

Table 1. Features of an information request.

Deriving Multiple Information Requests

The ability to derive multiple information requests from a user's notes is one of CENTAUR's unique features. When a user asks for information (i.e., pressing the Esc key), CENTAUR first identifies the current *input block*, a block of notes specifying the user's current information needs. If the user has never requested information before, the input block includes the entire notes entered so far. Otherwise, the input block starts with the top-most, unprocessed notes, and ends where the cursor is when the user presses the Esc key. In Figure 4(a), the input block starts with "*New York Hotels*" and ends with "*%Hilton*". After identifying the input block, CENTAUR concatenates all the text in the block into one big string. In Figure 4(a), the aggregated string is "*New York Hotels %Hilton*". Alternatively, users can also directly highlight a note block to be used as the current input.

Given an input string, CENTAUR first detects the boundaries of different information requests based on syntactic cues, such as common delimiters, shorthand notations, and indentations. Note indentations are first used to automatically separate the siblings into individual requests. In Figure 5, suppose that the siblings "*hotel class, avg customer rating*" and "*contact number*" were submitted together. In this case, they would form separate information requests. CENTAUR further checks whether delimiters like commas exist in the input. If delimiters are found, CENTAUR divides the input into multiple parts by the delimiters when appropriate[2]. For example, the input "*hotel class, avg customer rating*" in Figure 4(c) is broken into two segments, "*hotel class*" and "*avg customer rating*". Each segment is then further processed to form an information request.

For each identified segment, CENTAUR uses our previous natural language query interpretation engine to derive a set of semantic elements, including data concepts (e.g., *Hotel*), data attributes, and data constraints [13]. It also infers the semantic relationships between the elements, e.g., the relationship *located-in* between data concepts *Hotel* and *City*. More details of this work can be found in [13].

Above methods work reliably except in two cases where additional semantic rules are required to help correctly derive multiple information requests.

The first case is where a user's input is abbreviated. For example, the expression "*avg, min customer rating*" implies two information requests: retrieving the average and the minimal customer ratings. If CENTAUR used only syntactic cues (i.e., comma) to segment this input, it would produce undesired results (i.e., "*avg*" and "*min customer rating*"). To

2 CENTAUR does not always decompose an input into multiple pieces. For example, in constraint "*%Italian, rating>=4*", the delimiter is ignored, since it represents the "AND" of multiple restaurant constraints.

handle such cases, CENTAUR uses a set of rules. For example, one rule states that if there are multiple aggregation operators, such as *avg* and *min*, then each of the resulted segments must contain an aggregation operator and a data attribute. By this rule, CENTAUR can correctly divide the input "*avg, min customer rating*" into two pieces: "*avg customer rating*" and "*min customer rating*".

The second case is where a note input is ambiguous. For example, "*contact number*" in Figure 5 is ambiguous, since it may refer to the phone number or the fax number. To handle these cases, CENTAUR relies on a set of rules to automatically generate multiple information requests based on the nature of ambiguities. For example, one rule states that if an input (e.g., "*contact number*") implies multiple data attributes, then multiple information requests should be created for retrieving each of the identified attributes (e.g., "*phoneNumber*" and "*faxNumber*").

Deriving A Full Information Request in Context

As shown in Figure 4, a user's note input is often given in context. Accordingly, the current input block identified by CENTAUR may contain only partial information. For example, the input block identified in Figure 4(b) specifies an incomplete request ("*the cheapest hotel*"). To derive a full request (e.g., "*the cheapest Hilton hotel in New York*"), CENTAUR must integrate the interpretation of the current input with the existing note context.

Built on the context integration method in [10], ours extends it to exploit the structure of a note input. Here we describe our work in two steps: identifying integration candidates and determining integration operations.

Identifying integration candidates. *Integration candidates* are the information requests derived previously that can be used to help interpret the current input. By default, all information requests stored in a note context are potential candidates. Figure 6 shows two candidates: Request 1 and Request 2. Since not every stored request is relevant to the current input, CENTAUR uses the notes' structure to find suitable candidates. Our rationale is that note structural elements, such as indentations, imply the context in which a user input should be interpreted. Currently, CENTAUR uses two heuristics to look for two specific types of candidates based on the note structural information stored in their *input-location* and *indentation-level* features (Table 1).

The first heuristic is to look for candidates that represent the parent of the current input block so this block can be interpreted in the context of its parent. In Figure 4(d), the current input is "*%Marriott*". By the note structure, its parent is "*New York Hotels*". Accordingly, CENTAUR looks for the information request representing "*New York Hotels*".

Our second heuristic is to find candidates representing the siblings of the current input so it can be understood in the context of its siblings and their descendents[3]. In Figure 4(d), "*%Hilton*" is the sibling of the current input "*%Marriott*". CENTAUR thus seeks the information requests derived from the input "*%Hilton*" and its descendents.

3 In the case of having multiple siblings, currently CENTAUR considers only the sibling that is located right above the current input.

Operator	Description
add-link(R_1, R_2)	Link two nodes in requests R_1 and R_2 if there exists a semantic relationship in the ontology to link the two nodes (e.g., *Hotel located-in City*)
add-part(R_1, R_2)	Attach a data constraint or data attribute defined in R_2 to relevant data concepts in R_1 (e.g., attaching *roomRate* attribute to *Hotel* concept).
update-part(R_1, R_2)	Use the data constraint or data attribute defined in R_2 to update relevant items in R_1.
remove-part(R_1, R_2)	Use the data constraint (e.g., *chain=any*) defined in R_2 to remove the relevant one (e.g., *chain="Hilton"*) in R_1.

Table 2. Context integration operators used by CENTAUR.

If no integration candidates are found, CENTAUR then interprets the current input as is. In fact, users can use shorthand notation "\" to explicitly indicate stand-alone note input. For example, the input "\cheapest one" escapes any note context to request the cheapest hotels among *all* hotels.

Determining integration operations. After identifying integration candidates, CENTAUR selects *context integration operators* to merge the current information requests with the integration candidates. Table 2 lists the integration operators used by CENTAUR.

Operator selection is first based on the two possible types of relationships between an integration candidate and a current information request. An *add-part* operator is selected, if there is a *parent-child* relation; and an *update-part* operator is used for a *sibling* relation. Depending on the semantics of an information request, CENTAUR may use additional integration operators (e.g., *add-link* and *remove-part*).

Figure 7 shows the same text input but under two note structures. In Figure 7(a), by the note structure, an *update-part* operator is used to replace "*serving lunch, rating*>=2" with its new sibling "*rating*<=3". In contrast, in Figure 7(b), an *add-part* operator adds the new input "*rating*<=3" to its parent "*serving lunch, rating*>=2". Since the parent already contains a rating constraint ("*rating*>=2"), in this case, the two rating constraints are merged ("2<=*rating*<=3").

Assume that the new input is "*rating=any*" instead of "*rating*<=3" in Figure 7(b). CENTAUR would use operator *remove-part* to completely remove the rating constraint. However, if the new constraint were "*rating*>=3", it would then replace the old one ("*rating*>=2"). Moreover, suppose that the new constraint is a city constraint like "*%New York City*". CENTAUR would then use *add-link* operator to integrate the current information request on the city to constrain the locations of the restaurants.

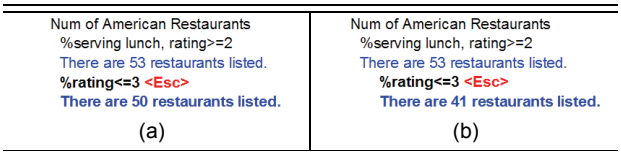


Figure 7. Input "*rating<=3*" under different note structures.

Adaptively Deriving New Information Requests

Unlike a traditional query-by-query information retrieval system like Google, CENTAUR allows a user to view his/her entire note input and corresponding retrieval results in a notepad at any time. As a result, a user may dynamically update any part of his/her notes (e.g., modification of existing notes or insertion of new notes) to express new or updated information needs. Depending on where the changes occur, such updates may require the re-interpretation of the *unchanged*, *previously entered* notes. For example, when "*contact number*" is inserted (Figure 5), the note context used to interpret the input "*%Marriott*" is also changed. "*%Marriott*" must then be re-interpreted in the new context to fulfill the user's updated needs.

To do so, CENTAUR first processes the user's new input to derive its corresponding information requests. For the new input in Figure 5, two information requests are derived: "*the phone number and fax number of the cheapest Hilton hotel in New York*". These two requests are then automatically inserted in the note context. Currently, CENTAUR assumes that all the notes located *below* the new input except the ones labeled as stand-alone notes are affected by the change. The affected section is then re-interpreted in the light of the updated context. In Figure 5, the re-interpretation of the affected input "*%Marriott*" yields two more requests: "*the phone number and fax number of the cheapest Marriott hotel in New York*". Consequently, Marriott contact numbers are also retrieved in addition to the Hilton's.

6. AUTOMATIC QUERY GENERATION

Given an information request, CENTAUR automatically generates one or more queries to satisfy the request. It does so in three steps. First, it identifies the number of queries and data sources needed. Second, it determines the parameters of each query. Third, it automatically formulates system-executable queries based on the data source(s) and query parameters determined in the first two steps.

6.1 Identifying Queries and Data Sources

As described in Section 5.1, an information request encodes the information that can usually be retrieved by a single query. However, such information (e.g., hotel reviews) may be stored in multiple data sources (e.g., relational databases vs. text collections). In such cases, multiple queries are needed to fulfill one information request.

To determine the number of queries needed for an information request, CENTAUR estimates the likelihood of each data source containing the requested information. A database is likely to supply the desired information if the data schema includes the same data concept(s) and data attribute(s) as those specified in the request. For text collections, we first extract relevant terms from the request. For example, we extract terms "*New York*", "*hotel*", and "*Hilton*" from Request 1 in Figure 6(a). We then use their frequencies in a text collection to estimate the likelihood of using this collection to satisfy the request [24].

6.2 Determining Query Type and Parameters

After identifying the number of queries and data sources needed for fulfilling an information request, CENTAUR then determines the query type and parameters for each query. It now supports two types of queries: SQL queries for database retrieval and keyword-based queries for text retrieval. Now the query type (e.g., SQL) is automatically determined by the data source (e.g., database) to be used.

To formulate an executable query, CENTAUR must also define necessary query parameters. Depending on the query type, the set of parameters may be different. For example, a SQL query requires proper data attribute(s) to be projected, while a keyword-based query demands a set of text terms. Generally, an information request already contains all the information for defining query parameters. For example, the data constraints specified in an information request can be directly mapped to the conditions in the WHERE clause of a SQL query. However, there are two cases where CENTAUR must perform additional work.

Determining Parameters for SQL Queries

One case is where an information request does not explicitly specify the exact information to be retrieved. For example, Request 2 in Figure 6(b) derived from the input "*cheapest one*" (Figure 4b) does not specify the exact hotel attributes to be retrieved (i.e., nothing to project in a SQL query). To obtain meaningful retrieval results, CENTAUR uses an optimization-based approach to data attribute selection [25]. This approach dynamically selects a sub-set of data attributes to retrieve by balancing a wide variety of selection criteria, including user preferences and data properties (e.g., data quality). In the case of above example, CENTAUR selects hotel name and room rate to retrieve (Figure 4b).

Determining Parameters for Keyword-Based Queries

The other case is where additional keywords are needed to augment the terms extracted from the current information request to formulate a keyword-based query. Similar to many existing query expansion efforts, CENTAUR includes additional keywords based on synonyms and morphological forms. For example, for the input "*customer reviews on location*" (Figure 8), CENTAUR includes keywords like "*location*", "*locate*", and "*place*".

If an information request does not specify what to be retrieved, CENTAUR automatically generates meaningful keywords to refine the query. Like selecting data attributes for a SQL query, CENTAUR selects a sub-set of attributes for keyword generation. In this case, it favors data attributes

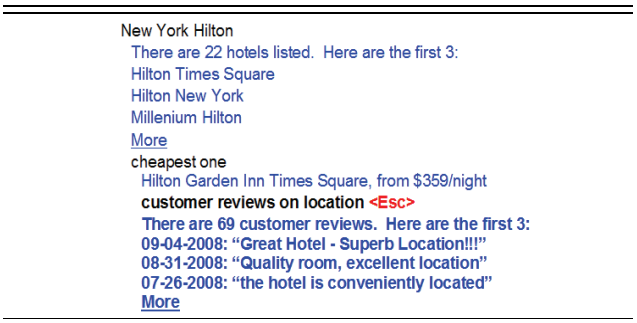


Figure 8. Generating additional keywords for better keyword-based text retrieval.

that appear more frequently in relevant text collections. Keywords are then generated for the selected data attributes.

Currently, CENTAUR reversely utilizes its dictionary that is used in the note interpretation to generate one or more keywords for expressing a data concept or data attribute.

6.3 Formulating Data Queries

After deciding the query type and parameters, CENTAUR uses a set of *domain-independent* query templates to automatically formulate system-executable queries. Each *query template* describes the required query parameters and a procedural method that uses the parameters to formulate a target query. CENTAUR now uses two query templates: one for generating keyword-based queries for Lucene text retrieval, and the other for formulating SQL queries for IBM DB2 database retrieval.

The Lucene-based query template allows the use of boolean operators (e.g., AND, OR) to combine multiple keywords in a query. The DB2-based query template defines various parameterized SQL fragments, mainly the SELECT and FROM statements and the WHERE clause. As described above, these parameters can either be directly extracted from an information request or dynamically determined by CENTAUR (e.g., projection attributes). The query formulation is mostly straightforward. However, due to the limitations of SQL itself, the use of nested SQL queries may be needed for complex requests, like "*Italian restaurants on the same street as Hilton New York*". Nonetheless, this is a pure programming effort, we do not discuss it here further.

7. EVALUATION

CENTAUR has been applied to two applications. One is a healthcare application where hospital physicians use notes to obtain relevant patient data in their daily summaries. The other is a hospitality application where users interactively use notes to gather relevant hotel and restaurant data. Due to the difficulty of recruiting medical professionals and the confidentiality of patient data, we have mainly evaluated CENTAUR in the hospitality application.

7.1 Data

We collected hotel and restaurant data from various sources and stored them in an IBM DB2 database. Our database includes both structured (e.g., hotel name and class) and unstructured fields (e.g., customer comments in text). A Lucene index was built from all the unstructured text.

7.2 Method

We designed and conducted a user study to evaluate the usefulness of CENTAUR in comparison with a baseline system. The baseline system (Q-GUI) provided users with a Google-like GUI. It allowed a user to enter one query at time in a search box. A user could enter a keyword-based query for text retrieval or a natural language query for database retrieval. Our original natural language query interpreter was used to process user-submitted queries [13].

We designed two similar but not identical information-seeking tasks. Each task required a user to answer 10 questions about hotels and restaurants that met a set of criteria (e.g., location and hotel class). To evaluate CENTAUR in a complex information-seeking task, we designed these questions such that subsequent questions were related to the previous ones. As a result, users must express their subsequent information needs in context. For example, four sample questions in one of the tasks were: (1) "*What is the lowest room rate of 3-star Hilton hotels in New York City*", (2) "*Is this rate higher or lower than the cheapest 3-start NYC Marriott hotel*", (3) "*What is the contact information of the cheaper one of these two hotels*", and (4) "*What is the contact information of the cheapest 3-star Hilton and Marriott hotels in White Plains*".

We recruited eight users, all of whom were familiar with the traditional query GUI but never used CENTAUR before. A within-subject comparison was used for our evaluation. Each user was asked to perform both tasks described above, one using CENTAUR and the other using the baseline system. To avoid potential biases such as learning effects, we randomly permuted the order of the task and system usage.

At the beginning of each user session, we gave a brief tutorial of both systems. Each user was allotted 30 minutes for each task. The user was asked to fill in an evaluation survey after each task. We logged the users' answers to all the questions. We also recorded all user interactions and the time used for answering each question.

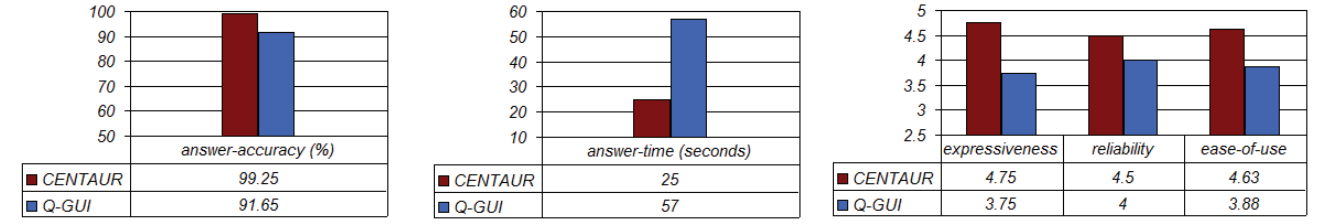
To compare the performance of the two systems, we used both objective and subjective measures. Our two objective measures were derived from the logs: *answer-accuracy* (percentage of correct answers) and *answer-time* (the time taken to obtain the answer to a question). For each system, we computed the average *answer-accuracy* and *answer-time* across all the users and questions. We also extracted three subjective measures from the user surveys: *expressiveness* (how easily a user could express his/her needs), *reliability* (how accurately a user's needs were understood), and *ease-of-use* (how easy it is to use the system for the task at hand). All the subjective measures were rated on a 5-point scale, with 1 being the worst and 5 being the best.

7.3 Result Analysis

We examined both objective and subjective data collected from our study. As shown in Figure 9, CENTAUR outperformed the baseline system across all the metrics. Objectively, CENTAUR helped users gather information to answer the questions more accurately (Figure 9a) and in less time (Figure 9b). Subjectively, all the users preferred CENTAUR in terms of its expressiveness, reliability, and ease-of-use (Figure 9c).

The differences are significant in two aspects: CENTAUR helped users produce an answer significantly faster ($p<.01$) with a significantly better accuracy ($p<.02$). Compared to the baseline system, CENTAUR helped reduce the *answer-time* by 17%–65% and increase the *answer-accuracy* by 6%–19%, across all the users. Our analysis of the user interaction logs attributed CENTAUR's efficiency and accuracy to three main factors.

First, specifying information needs in brief notes was faster than manually formulating multiple keyword-based or natural language queries. Second, a user could view his/her



(a) Comparing average answer-accuracy (b) Comparing average answer-time (c) Subjective measure comparison

Figure 9. Study results comparison between CENTAUR and the baseline by objective measures (a-b) and subjective measures (c).

entire notes and edit any part of the notes to easily express his/her new or updated requests. Moreover, CENTAUR could automatically infer a user's implicit needs to adaptively retrieve needed information (e.g., Figure 5). This saved a user's time for explicitly expressing every request. Third, CENTAUR provided the users with more relevant results in text retrieval. It did so by supporting effective query expansions based on the meanings of the note input.

The subjective data indicated that the users felt that they could easily express their information needs to CENTAUR using interactive notes. On the other hand, CENTAUR could correctly interpret their notes to provide them with needed information. Although both systems supported natural language queries, the better reliability of CENTAUR was attributed to two factors: (1) Most of the notes were briefer and less ambiguous (especially with the use of shorthand notations) than the queries submitted to the baseline; (2) The use of indentations in notes helped define the correct interpretation context, while the users could not explicitly indicate a query context when using the baseline system. Both factors helped CENTAUR reduce ambiguities when interpreting user notes in context, which in turn improved its reliability.

The users' comments were consistent with our analysis. All the users gave positive comments regarding the key features of CENTAUR. Specifically, six out of eight users thought the best feature of CENTAUR was its notepad interface, where they could conveniently express their requests *and* receive information in one place. Moreover, they liked the flexibility of incrementally gathering one piece of information at a time or obtaining multiple pieces of information together. Seven users considered it very useful for CENTAUR to automatically detect their explicit and implicit needs in context and adaptively retrieve desired information (e.g., Figure 4d and Figure 5).

The users also provided valuable feedback on where CENTAUR could be further improved. Most of the users desired some guidance and confirmation during their note input. Since they were accustomed to the limited functionality provided by a traditional query GUI, there was a sense of uncertainty on the form of notes allowed. Several users also suggested better grouping and presentations of the retrieved information, especially when multiple data items were retrieved (e.g., a list of hotels).

8. DISCUSSIONS

During the development and evaluation of CENTAUR, we have learned valuable lessons and received feedback on various aspects of CENTAUR. Here we share some of the feedback and discuss the remaining challenges.

System Portability

One of the main concerns is the portability of CENTAUR, since it uses various types of semantic information (e.g., data ontology) to interpret a user's notes. From our own experience, it normally takes three main steps to port CENTAUR to a new application.

First, we define a data ontology to include the data concepts and data attributes specific to the application domain. Currently an ontology is semi-automatically created based on a database schema. Data tables and columns are automatically mapped to data concepts and attributes, respectively. The relationships between data concepts are extracted either based on table relations (e.g., foreign keys) or based on manual annotations. We deliberately create a "*shallow*" ontology to facilitate portability. Second, we create a dictionary that associates words/phrases with data elements in the ontology. The dictionary can also be semi-automatically populated by mining databases to extract the needed information (e.g., all columns and their respective values). Third, if the new application requires the use of unstructured data sources such as text documents, these data sources may need to be pre-processed (e.g., indexed).

Since both the ontology and dictionary are used as knowledge sources to interpret a user's input, CENTAUR can handle a new ontology or dictionary without changing its interpretation algorithm. As a result, it can be easily extended to cover new data concepts and data attributes defined in an ontology or dictionary. In addition, we can easily extend CENTAUR to cover new query formats by adding new query templates (e.g., a template for Oracle SQL query or for text query using Google APIs).

Exploiting Note Structure

Unlike *ad hoc* user input, a user's notes often bear a certain structure. Although CENTAUR has used several syntactic cues (e.g., shorthand notations) in its note interpretation, we believe that it could further exploit the note's structural information. For example, a user may use brackets to indicate different note blocks, which help CENTAUR to segment different information requests without even using any semantic inference. In addition, the use of syntactic cues in

note interpretation makes CENTAUR more portable to different domains and applications. However, asking users to provide additional syntactic cues may increase user burdens. The challenge is to balance the burdens on the system (in interpretation) and on the users (in expression).

Query Performance Optimization

Currently, we have limited the information content allowed in a single information request to simplify query generation. However, this may negatively impact the system performance especially when data volume is large (e.g., terabytes). For example, two information requests are used to represent the retrieval of the minimal and maximal room rates of a hotel set, since now each information request is limited to a single aggregated data attribute. Accordingly, two SQL queries are needed to retrieve the information. Executing two such queries would be time consuming if the underlying data set is massive. To achieve the desired performance, query optimization may be needed. In certain cases, multiple queries may be merged into one. However, it may be difficult to decide whether and how multiple queries can be combined. For example, it is non-trivial for CENTAUR to dynamically determine the optimal number of queries needed for retrieving the count, hotel class, and average customer rating of the same hotel set.

9. CONCLUSIONS

In this paper, we present an interactive, smart notepad system, called CENTAUR, which retrieves the desired information directly based on a user's note input. We have focused on CENTAUR's two key aspects. First, we explain how it dynamically interprets a user's notes in context to derive a set of information requests. Specifically, we adopt and augment a semantics-based natural language query interpretation approach to accurately interpret notes. Our work can identify multiple information requests at once, derive a full information request in context, and automatically re-evaluate existing notes based on note updates. Second, we describe how CENTAUR automatically generates queries to satisfy the derived information requests. In particular, CENTAUR uses a hybrid of rule-based and procedural approach to determine the number of queries needed, decide proper query parameters, and format target queries.

As a result, CENTAUR provides users with three distinct benefits. First, users can focus on *what* to retrieve instead of *how*. Second, users can efficiently fulfill multiple information requests at once and easily obtain information stored in different data sources. Third, users can easily update their notes to express their new or updated information needs and automatically obtain new information. Our preliminary study results also demonstrate that CENTAUR effectively assists users in their information-seeking tasks.

REFERENCES

[1] J. Allen, G. Ferguson, and A. Stent. An architecture for more realistic conversational systems. In *IUI '01*, pages 14–17.

[2] M. Baldonado and T. Winograd. Sensemaker: an information-exploration interface supporting the contextual evolution of a user's interests. In *CHI '97*, pages 11–18.

[3] D. Billsus, D. Hilbert, and D. Maynes-Aminzade. Improving proactive information systems. In *IUI '05*, pages 159–166.

[4] J. Budzik and K. Hammond. Watson: Anticipating and contextualizing information needs. In *of AAIS' 99* pages 727–740.

[5] D. H. Chau, B. A. Myers, and A. Faulring. What to do when search fails: Finding information by association. In *CHI '08*, pages 999–1008.

[6] S. Dumais, E. Cutrell, R. Sarin, and E. Horvitz. Implicit queries (IQ) for contextualized search. In *SIGIR '04*, pages 594–595.

[7] S. Gauch, J. Chaffee, and A. Pretschner. Ontology-based personalized search and browsing. *Web Intelligence and Agent System*, 1(3-4):219–234, 2003.

[8] D. Goncalves and J. A. Jorge. In search of personal information: Narrative-based interfaces. In *IUI '08*, pages 179–188.

[9] K. Gyllstrom and C. Soules. Seeing is retrieving: Building information context from what the user sees. In *IUI '08*, pages 189–198.

[10] K. Houck. Contextual revision in information-seeking conversation systems. In *ICSP'04*, pages 201–204, 2004.

[11] M. Johnson, S. Bangalore, G. Vasireddy, A. Stent, P. Ehlen, M. Walker, S. Whittaker, and P. Maloor. Match: An architecture for multimodal dialogue systems. In *ACL '02*, pages 376–383.

[12] R. Kraft, F. Maghoul, and C. Chang. Y!Q: contextual search at the point of inspiration. In *CIKM '05*, pages 816–823.

[13] S. Pan, S. Shen, M. Zhou, and K. Houck. Two-way adaptation for robust input interpretation for practical multimodal interaction. In *IUI '05*, pages 25–32.

[14] A. Popescu, O. Etzioni, and H. Kautz. Towards a theory of natural language interfaces to databases. In *IUI '03*, pages 149–157.

[15] Y. Qu. A sensemaking-supporting information gathering system. In *CHI '03*, pages 906–907.

[16] B. Rhodes and P. Maes. Just-in-time information retrieval agents. *IBM Systems Journal*, 39(3-4):685–704, 2000.

[17] J. Shen, W. Geyer, M. Muller, C. Dugan, B. Brownholtz, and D. Millen. Automatically finding and recommending resources to support knowledge workers' activities. In *IUI '08*, pages 207–216.

[18] X. Shen, B. Tan, and C. Zhai. Context-sensitive information retrieval using implicit feedback. In *SIGIR '05*, pages 43–50.

[19] X. Shen and C. Zhai. Exploiting query history for document ranking in interactive information retrieval. In *SIGIR '03*, pages 377–378.

[20] M. Speretta. Personalizing search based on user search histories. In *CIKM '04*.

[21] J. Teevan, S. Dumais, and E.Horvitz. Personalizing search via automated analysis of interests and activities. In *SIGIR '05*, pages 449–456.

[22] E. Voorhees and H. Tang. Overview of the TREC 2005 question answering track. In *TREC '05*.

[23] Z. Wen, M. Zhou, and V. Aggarwal. Context-aware adaptive information retrieval for investigative tasks. In *IUI '07*, pages 122–131.

[24] J. Xu and W. B. Croft. Cluster-based language models for distributed retrieval. In *SIGIR '99*, pages 254–261.

[25] M. Zhou and V. Aggarwal. An optimization-based approach to dynamic data content selection in intelligent multimedia interfaces. In *UIST '04*, pages 227–236.