GUI— Phooey! : The Case for Text Input

Max Van Kleek[1], Michael Bernstein[1], David R. Karger[1], mc schraefel[2]
[1]MIT CSAIL, 32 Vassar Street, Cambridge MA 02139
[2]USouthampton, Southampton, UK, S017 1BJ
{emax, msbernst, karger, mc}@csail.mit.edu

ABSTRACT
Information cannot be found if it is not entered. Research shows that existing rich graphical application approaches interfere with user input in many ways, forcing complex interactions to enter simple information, requiring complex cognition to decide where the data should be stored, and limiting the kind of information that can be entered to what can fit into specific applications' data models. Freeform text entry suffers from none of these limitations but produces data that is hard to retrieve or visualize. We describe the design and implementation of *Jourknow*. a system that aims to bridge these two modalities, supporting lightweight text entry and weightless context capture that produces enough structure to support rich interactive presentation and retrieval of the arbitrary information entered.

ACM Classification: H5.2 [Information interfaces and presentation]: User Interfaces. - Graphical l user interfaces.

General terms: Design, Human Factors

Keywords: Personal Information Management, input, text, structured text, lightweight input

INTRODUCTION
Too often, even the best information retrieval tools cannot help us find what we are seeking, because the information we want was never recorded. This can happen for many reasons. Sometimes, we simply do not recognize that the information might be needed later [18]. At other times, the perceived cost to launch and navigate through multiple applications to capture the information seems too high for the currently perceived value of the data. Finally, our strong desire to record some information can be stymied by the fact that there is no natural place for it---no folder where we have confidence that we will be able to find it when we need it [7], or no native application that may be associated with the particular kind of data being entered.

Many of these problems vanish if we turn to a much older recording technology: text. Recording a fragment of text simply requires picking up a pen or typing at a keyboard. When we enter text, each (pen or key) stroke is being used to record the actual information we care about---none is wasted on application navigation or configuration. The linear structure of text means there's always an obvious place to put anything---at the end. And the free form of text means we can record anything we want to about anything, without worrying whether it fits some application schema or should be split over multiple applications. All of this means that we have to do less to record text, which makes it more efficient and also less of an interruption and distraction than using complex applications.

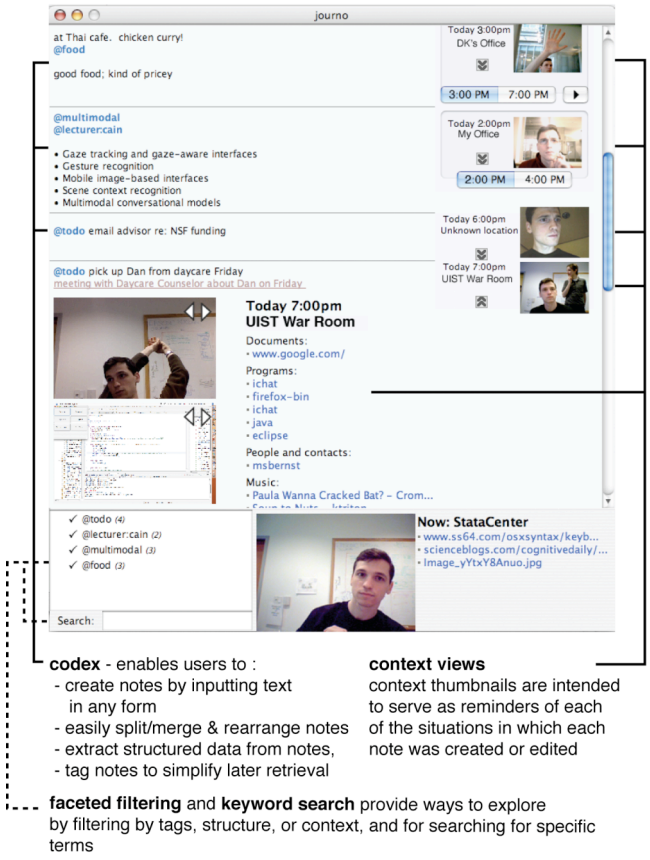

Figure 1: The Jourknow interaction interface displaying the set of textual notes, which we collectively refer to as the *codex*. In addition to the text, Jourknow's side panels provide facilities for quickly scanning notes through *episode thumbnails* and navigating non-linearly by filtering notes according to facets that characterize their contents. The *current context display* (bottom right) provides a preview of how the current editing session will be later portrayed in episodes.

While text is an outstanding solution for recording information, its weakness lies in retrieval. Text's fixed linear form reduces us to scanning through it for information we need. Even with electronic text, the lack of structure means we cannot filter or sort by various properties of the information. When we aren't sure what we want, a blank text search box offers few cues to help us construct an appropriate query [34]. The shorthand we use to record information in a given context can make it both hard to find and incomprehensible when we return to it later without that context [19]. Furthermore, only text we explicitly enter is re-

corded, without any of the related contextual information (such as a timestamp) that might be known to a sophisticated application.

In this paper we argue that it is possible and desirable to combine the easy input affordances of text with the powerful retrieval and visualization capabilities of graphical applications. We present *Jourknow*, a personal journal that "knows what you mean" when you write things. That is, it supports lightweight text input to be used for capturing richly structured information for later retrieval and navigation in variety of interfaces. *Jourknow* provides the following facilities:

- Entry of information by typing arbitrary scraps of text (with all the text-input benefits mentioned above)
- Inclusion of structured information in the text through a simple and user-extensible shorthand we call "pidgin"
- Extraction of structure from scraps of text, identifying entities and relationships between them
- An arbitrarily flexible data model to record whatever structure a user considers important
- Association of automatically-measured context with the information being recorded
- Search and faceted browsing based on tags, entities, and relations for finding relevant text scraps
- Automatic routing of relevant pieces of the entered information to traditional PIM applications such as calendar, address book, and web browser so that it can be retrieved and visualized using those domain-specific tools.

In order to deliver these interactions, we had to solve several challenges: capturing structure from text not entered in a form, modeling capture of desktop state for appropriate association with a scrap, supporting interpretation and retrieval of individual text scraps, and integration of captured data for use with existing applications. In the following sections we present the related work that informs our approach, describe the interaction design and implementation design details. We then discuss our immediate plans for extending the Jourknow platform, iterating its design, and most importantly, studying the impact of Jourknow on people's information scrap entry and reuse behavior.

RELATED WORK

In personal information management (PIM) research, there is a tension between the desire for lightweight mechanisms when inputting data and the desire for that data to be richly structured once it has been entered. We see this tension expressed in Kalnikaite *et al.'s* recent work in personal note-taking tools, which describes the need for both efficiency (reducing the effort required to add notes with a particular tool) and accuracy (the fidelity of the resulting notes and their later utility) [17]. Their study revealed that efficiency was key to determining whether a user would choose to use a particular tool for capture, but accuracy determined whether notes would be revisited, and how long notes would continue to be useful after they were captured. There was little evidence that one tool regularly supported both attributes. Our goal is to bridge this gap.

Challenges of structured input

For initial data capture, the need for fast, lightweight input was driven by an understanding that both lack of control over interruptions [8] and the cognitive load of multitasking [14, 24] could severely impact performance of carrying out a given task. Ross and Nisbett [29] identified some of these impediments as *channel factors*, "small but critical facilitators" that could dramatically impact a person's actions. Channel factor analysis has identified how even simple intervening steps to a task, such as, for example, needing to explicitly assign a name to a file at time of its creation, or to navigate a complex set of GUI widgets on an input form, impede a user's work flow, occasionally to the point of derailing them from ultimately accomplishing their task(s).

Regardless of the UI, the mere requirement for structure can interfere with peoples' ability to record the information they want. For instance, primary work by Bellotti et al. [3] and Blandford [6] each found that people naturally use partial, incomplete, or often vague descriptions of data in their personal notes, while PIM tools often require rigid, formal and exact specifications. According to Kalnikaite [17], this mismatch between human practice and machine form creates a considerable barrier to PIM tool use.

The need for structure

Despite our resistance as users to current UI mechanisms like form-filling to associate structure with information as found in calendar and address book applications, without that structure, information becomes difficult to utilize efficiently. Indeed, barring a word processor, applications from iTunes to spreadsheets to large scale PIMs like Lotus Notes, to databases, to faceted browsers [12] have well-defined structured models to drive their user interfaces; without them these tools would simply not work. Structured data is the *lingua franca* of these applications. These structures can then be mined to support the user's needs, such as through activity management [11, 18, 26], and data manipulation and transformation [19, 31] Unfortunately, the tremendous information management benefits offered by rich graphical user interfaces over richly structured data are entirely lost if the cost of entering information into those applications deters people from doing so. [3]

Information Extraction

The field of *information extraction* (IE) [25] has focused on the general problem of automatically identifying and interpreting structured information in free text, including references to things and relations among them. Much of the work in IE, however, has surrounded analyzing either proper well-formed written or transcribed spoken *natural language*, which, as we have found [5] differs in form significantly from the abbreviated, often ungrammatical forms people seem to use in their information scraps. Since techniques for mining information from these abbreviated forms (which often contain less syntactic and grammatical structure) are significantly less well understood, we have had to adapt existing NLP extraction techniques, choosing particularly simple approaches in order to ensure predict-

able and reliable behavior. This is described in the Implementation section.

(Re)use of data/structure: contexts

In order to help recover entered data, Jourknow has also been informed by studies of remembrance habits, which find that data is often indexed by information extrinsic to the bits themselves, such as relevant people [9], the path to the information, or temporal aspects [22, 28]. There is evidence to support the use of such virtual markers: Sellen et al. has shown that even arbitrary pictures automatically taken at intervals by a camera strapped to the user's body can effectively prime a person's memory to help them recall specific events in their daily lives [33]. Recently, Whittaker observed that automatically capturing and associating information with notes (such as audio records) can help people remember facts and meanings associated with personal notes, which otherwise fade within a month [17]. Such "life logging" has also been explored in the personal notebook space. Dumais et al.'s Stuff-I've-Seen [9] demonstrates that context pertaining to when users last viewed documents can be used to ease their re-finding. *Pepys* [21] and the *Remembrance Agent* [27] incorporate location and the identity of nearby persons to retrieve notes created in similar contexts. *MyLifeBits* from MSR has also sought to build a lifetime personal information store that computes spatial and temporal correlations among resources for retrieval tasks, such as searching for documents based on co-occurring events, co-located items, and time of access [10].

INTERACTION

In this section, we describe the design of our system, as pictured in Figure 1. The overarching goal is to create an effective bridge between rapid, low-cost, unconstrained input based on text and context and effective retrieval and output based on a structured data model and rich GUI. Our design achieves this end through the following means.

Unconstrained text entry. Information is entered as free text in a small text widget that can be invoked anywhere by a keystroke.

Entity recognition. Jourknow recognizes named entities in the entered text and associates the entered information with them. New entities can be defined as needed.

Structure recognition. Jourknow uses simple subject-oriented grammars (or *pidgins*) about defined concepts such as meetings, dates, and locations for parsing structured information in text fragments; *e.g.* "*meeting with joe at 5pm in G592*", or "*add milk to shopping list*". Users can change what forms Jourknow recognizes by example, as described later, including extending the grammar to handle new types of information, such as, for example, names and properties of stamps in a user's collection.

Context capture. Jourknow watches what a user is doing and where they are doing it, and records that information with input text to assist in later retrieval.

Application integration. Jourknow provides access to entered text containing information that aligns with structured applications directly through these applications, such as calendar, address book, and web browser, so that the information can be used easily when they are needed, and can be organized using those rich domain-specific tools.

First-class text fragments.. Jourknow treats its input, not as one big blob of text, but as a large collection of text scraps and entities that can be retrieved individually. This offers a much finer grain of retrieval than other system, so that user can home in on exactly what they need.

Structured retrieval. Jourknow offers search and faceted browsing based on tags, entities, and relations for finding relevant text scraps and entities.

In the remainder of this section, we offer more detail on the mechanisms listed above.

Text: Lightweight Input of Information

The two main design goals for data input into Jourknow are, first, *efficiency*, the ability to get the information into the computer as easily and conveniently for the user as possible; and second, *fidelity,* the facility to capture the structure of information in as flexible and non-restrictive a way as possible.

To address light-weight data entry, Jourknow provides a simple unrestrictive text input in which the user may type notes in any way they please. Specifically, data entry into Jourknow divides the user's text buffer into notes that may be freely rearranged. The set of all a person's notes and their contents in Jourknow is called the *codex*. To reduce the cost of switching applications to add notes to Jourknow, Jourknow provides a number of shortcut hotkeys that make it possible to interact with it while other applications have focus. The "toggle visibility" hotkey instantly brings Jourknow into focus, and dismisses it from view when pushed a second time. A "paste" hotkey sequence sends any selected text from any application directly into a new Jourknow note and brings it into view, while a "bookmark" hotkey, if pressed while viewing any document or web page, adds the document's title and link to a new Jourknow note. Similarly, text may be simply dragged and dropped or pasted into an existing or new note from another application, rearranged within a note, or moved between notes.

Notes in the codex can be categorized by adding tags, which by default are identified syntactically as a single word starting with the '@' symbol. Notes may have any number of tags. Additionally, tags may optionally have values, which by default are identified with a colon following a tag. For example, class notes for an algorithms course could be tagged just as "@**class**", both "@**class**" and "@**algorithms**", or alternatively, "@**class:algorithms**". These tags may be later used to quickly select subsets of notes to be viewed, as described in Filtering, Exploration, Finding and Reminding. Tags are scoped to the entire note, and thus all text and *subtext* (described next) in the note inherit the tag. The syntax used to recognize tags may be changed in Jourknow's preferences.

Beyond unstructured text and tagging, Jourknow provides two mechanisms by which users can express structured information to the system in a way that Jourknow will be

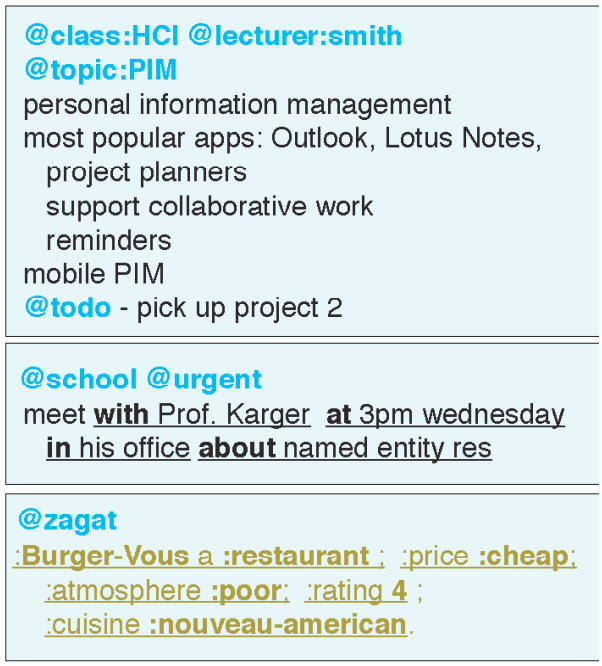

Figure 2. Jourknow supports combinations of tags and unstructured text (top), pidgin grammars (middle), and Notation3 (bottom).

able to understand. The first is a simplified language or "pidgin" [32] which allows users to express structured data items more naturally for particular predefined domains, such as for events, meetings, or address/contact information. Our goal is for our pidgin to automatically capture common kinds of structured data entry that are entered in unstructured form. The second mechanism is the use of a lightweight triple syntax (based on Notation3 [4]), a domain-generic grammar with which the user may make statements to express arbitrary structural properties and relationships among entities, without having to predefine pidgin for the domain or ontology. Examples of each are illustrated in Figure 2. When Jourknow's pattern recognizers (described in System Implementation) identify either type of structured information, Jourknow creates subtext for it, which are structured entities that reflect its interpretation of what was written.

If a user wishes Jourknow to recognize new pidgin phrases for new domains not covered by the Jourknow's base grammars for events, addresses and to-dos, Jourknow makes it easy to do so. The user can simply use a pidgin *means* expression anywhere in their codex, which, when interpreted, creates a subtext structure (described in the following section) representing a mini-grammar that the pidgin language processor can subsequently use to recognize the new forms. An example of a means expression to handle pidgin forms such as "new cafe Diesel at Davis Sq wifi yes" into subtext would be "'*new cafe <A> at <B> wifi <C>' means* ***<A> a :cafe; :at_location <B>; :has_free_wifi <C>*** ." , where the expression after "means" represents the intended structure expressed in Notation3, and tokens in brackets represent corresponding "slots" for values. Users can also require that the slots conform to a particular type by adding the required type in brackets, for example: "**<A Date>**".

Subtext: the structure within the codex

The subtext consists of instantiations of any structures described in the codex. Once instantiated, these structures can be "brought to life,"---manipulated like objects in traditional PIM applications, used to set reminders, or sorted and filtered by property or value. To maximize their availability and utility, Jourknow exports a view of subtext structures that represent PIM data types such as events, contacts, and to-do items to the user's standard PIM applications. Edits via these external representations are reflected in the subtext, and are made visible in the codex through a revision indicator. We describe how this is done in System Implementation.

For example, the pidgin "*mtg at Luna Cafe @ 5pm w/ Akemi cell 617-xxx-yyyy re:camping this weekend"* translates to a subtext which encodes the fact that there is a meeting that is happening at a location known as the "Luna Café", at 5pm today, with a person named "Akemi", whose cell phone number is "617-xxx-yyyy", on the topic of "camping this weekend". Instantiation of this subtext causes an event to appear in the user's calendaring application with the appropriate date, time and subject, as well as contact information to appear (if one didn't already exist) for a person named "Akemi" with the appropriate phone number, in the user's address book.

Note that ambiguity with resolving names of places and people to records is in general a significantly difficult problem. For example, if the user already has several contacts with a name "Akemi", it would be impossible for Jourknow to tell (from the example above), which Akemi the user was referencing. Jourknow deals with this by first attempting to find an exact match for an identifier among the names of all the entities in the subtext. If an exact match is unavailable, Jourknow presents a list of near misses with edit distances less than 20 percent of the identifier's length. If the user does not choose one of the presented alternatives, Jourknow resorts to creating a new subtext item to represent the structure being mentioned.

Figure 3. Clicking on text which has associated subtext displays the *structure editor*, a form-like view of the subtext structure that allows the user to quickly verify and update values for properties, similar to a traditional form-based user interface

Such ambiguous naming situations can be avoided in several ways. First, the user can add nicknames to various entities by adding a nickname pidgin expression, which creates a subtext structure that causes the nickname to be functionally equivalent to the original form when matching. For example, "*AK means Akemi Kuromasa*" generates a subtext that establishes the phrase "AK" as equivalent to "*Akemi Kuromasa*" when searching for a subtext element. Users can add additional names explicitly to specific subtext entities through either pidgin or Notation3 expressions, such as ":**AK :name Akemi-chan.**", which is a Notation3 expression attaching the name *Akemi-chan* to the subtext entity identified as :**AK**, causing any subsequent references to *Akemi-chan* in any pidgin expressions to resolve to **:AK**. More details on named entity matching are described in System Implementation.

As soon as a structured data item is recognized, Jourknow provides visual feedback of how the expression was interpreted, by indicating groupings of words into clauses as parsed, boldfacing headwords, and underlining values for recognized data types (see Figure 2). This feedback is important because as described later, many pidgin grammars are ambiguous; therefore, it may not be possible for Jourknow to select the correct parse. If Jourknow chooses an incorrect parse, the user may hit a button to cycle through alternative parses, or failing that, correct the interpretation manually. Similarly, this feedback gives the opportunity for the user to see what entities Jourknow has matched in the pidgin expression, if any, and to allow the user to correct its name resolution if necessary.

Clicking on any entity that Jourknow has recognized in a text fragment conjures a view of the subtext known as the *structure editor*, which allows the user to directly view and manipulate a subtext element's properties and values (Figure 3). When the user completes editing of the subtext, Jourknow then updates the text to reflect the user's changes to ensure that the codex always maintains a correct view of the subtext.

Context: the activity and environment of the data

The context consists of information describing the circumstances under which a particular note was created or edited. The purpose of this information is to be useful as a "hook" to help quickly re-find a piece of text, or for reminding the user about the text's meaning, by priming recall of the actual situation in which the note was written. Specifically, Jourknow captures two types of situational context around the time a piece of text was edited: the user's desktop activities, and aspects of their physical environment. An example of the former are files and web pages the user was examining or editing, applications the user was using, and with whom the user was communicating. Examples of the user's physical context include their location, the identity of people detected nearby, and photos of the user from the user's laptop's web cam. Our goal is to make these available geo/temporal/activity contexts serve in the same way that physical contexts help re-finding of physical notes.

We make the captured context visible to the user by means of episodes attached to each note (Figure 4). Intuitively, an episode should correspond to a continuous period of time that was characterized by one activity. Jourknow has the ability, however, to segment time according to different criteria, to support the multitude of ways we think about our activities. Example segmentations involve simple time-based approaches such as by hour, time of day (morning/afternoon/evening) or those defined according to particular phenomena observed in the context, such as the user's location, times during which certain music was being played, or stretches of uninterrupted activity in a particular application. For each episode that such a segmentation defines, Jourknow consolidates and summarizes all the context observations that intersect with the time interval for which that episode was defined, into an episode context view (Figure 4).

To explore the detailed captured context of a Jourknow entry, the user can view a selection of the most relevant information surrounding the writing on the note by expanding the context panel attached to each note. This view includes desktop screenshots, photos of the user and his or her surroundings, the most active documents and web pages, and location information (including an interactive map). If the user edited a note over several different sessions, a tabbed interface allows the user to view all of the relevant gathered contexts.

Filtering, Exploration, Finding and Reminding

There are various ways a person may wish to access information once it has been captured in their codex. For example, when in class, a student may wish to "focus" on a subset of their codex consisting of all the notes for that particular class, ordered chronologically, or by topic. Therefore, users may wish to filter or order notes based on some aspect of their content, such as those tagged a certain type, containing certain text or entities, or a piece of structured data. A user may also wish to select subsets of notes that meet criteria concerning the situation(s) in which they were created, viewed, accessed or edited, such as being edited at a particular place or time. In Jourknow, we sought to facilitate both types of criteria for filtering, in order to facilitate focusing on a particular subset of notes, exploration, and re-finding. Using data contained in the text, subtext,

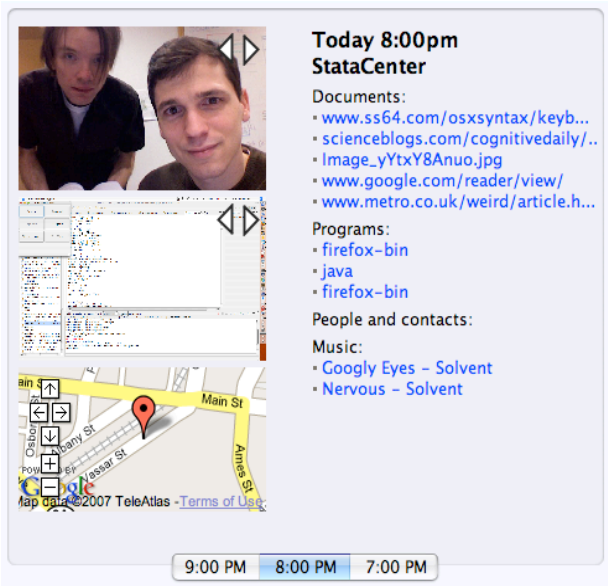

Figure 4. The expanded view of context summarizes the user's activities during the time interval specified by the episode. The salience heuristics used to select events to be shown to generate the summaries are described in section *Implementation*. Tabs at the bottom allow users to see past episodes for notes that have been edited in multiple episodes.

and context, and the correspondences that Jourknow maintains between them, it became possible to unify all of these different axes for filtering into a single, simple interface. For Jourknow, we have chosen the mechanism of *incremental search* that combines *faceted browsing* [12] with keyword search. Facets allow the user to quickly filter notes based on either content- or context-based features or combinations thereof, such as tags, tags with values, structure therein contained, access time, location, by associated resources. Users first select the facet with which they wish to filter their codex, and then are presented list of values they can select from for that facet. Keyword search lets you further refine or directly jump to notes containing a particular word or phrase.

After the desired subset of notes is achieved, using episode views it becomes possible to explore relationships among the contextual factors surrounding a note to answer questions such as: *where was I exactly and what was I doing exactly when I came up with those great ideas?*

Finally, beyond exploring information in an ad hoc way, the user may wish to set up information to surface to the foreground of the desktop in a certain situation in the future, such as "*tell mc about this when i'm in her office*." Bellotti describes this kind of information foregrounds as supporting the "in the way property" of reminding [3]. Jourknow's tagging and pidgin enables these kinds of reminders to be triggered currently by any part of the context such as time, location or application state.

SYSTEM IMPLEMENTATION

The key technical challenges surrounded supporting the following functionality:

1. Unconstrained textual input
2. Flexibility in how information can be structured; without requiring people to predefine (or adhere to predefined) ontologies
3. Interface with desktop applications; specifically, alignment of subtext with applications' data ontologies
4. Extraction of subtext from unconstrained text; particularly supporting incomplete grammatical input, partial phrases, and informal language
5. Capture of context, and subsequent selection and presentation of relevant contextual events for supporting effective re-finding and memory priming
6. Maintaining appropriate correspondences among the user's text, extracted subtext, and captured context

In this section, we describe our solution to these six challenges; Figure 5 illustrates the design of Jourknow.

Data Model: Three Representations

Text

Jourknow maintains three different knowledge bases (KBs) to hold the structures that become the text, subtext, and context components introduced earlier. The text KB maintains what is traditionally thought of as the "contents" of the codex, specifically what is needed to construct each of the notes and their respective contents. Instead of saving and overwriting snapshots of the notes as most text editors do today, the text KB preserves the entire history of edits the user made, similar to log-structured file systems [16]. Maintaining a complete edit history in this manner enables Jourknow to identify exactly when each character was created, edited or automatically generated in response to an external edit to underlying subtext, as well as to recreate the state of the journal at any arbitrary point in the past. Being able to identify the time of creation efficiently for each character in the codex is critical to Jourknow's functionality, because it is the key by which Jourknow establishes a correspondence among text, subtext, and the context chronology. To facilitate fast creation time lookup of characters, Jourknow maintains (in memory) a data structure which packs the character creation time (represented to millisecond granularity) into the data structure used to represent each character in the buffer. Note that there is no scalability issue here, as the complete set of text typed by an individual cannot fill an appreciable portion of current memory. This creation time is kept with the character for the entire duration of its existence, and follows each character as it moves due to edits to surrounding text. New characters assume the current time, and characters that are cut or copied from one location in Jourknow and pasted elsewhere gain additional timestamps corresponding to each paste. This makes it possible to "manually re-plant" context associated with a note simply by moving appropriate text between notes. These time signatures of text are used to establish correspondences with subtext and context elements, described in the next two sections.

Subtext and application integration

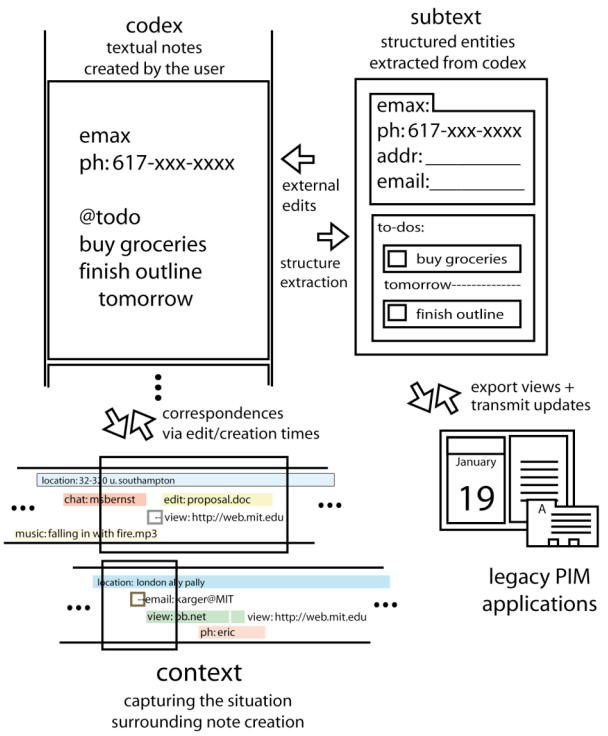

Figure 5. Architecture of Jourknow, illustrating the relationship between the codex, subtext, and context.

Entities in the subtext KB are represented as graphs in RDF [13]. Subtext that originates from pattern extractors such as the pidgin parser or the Notation3 processor are grounded in the PLUM and Jourknow ontologies which are mapped to standard RDF ontologies, such as iCalendar [7] for events and vCard [15] for contacts. Use of these standard ontologies makes it possible to use existing tools [23] to help with ontology alignment to schemas used by external sources, simplifying the process of importing subtext such as events from atom/RSS feeds, contacts from the user's LDAP server or IMAP e-mail account, and bookmarks from the user's web browser. Importing subtext items can assist the name ambiguity problems mentioned earlier by providing additional information with which to identify people, places and things mentioned in the codex.

Similarly, when manifestations of the user's subtext ("shadows") are exported to the user's applications, ontologies must first be aligned with the target application's schema. Unfortunately, establishing a good mapping from instances grounded in the rich, expressive ontologies of the semantic web, to rigid schemas of PIM applications often requires somewhat arbitrary decisions regarding determining which fields best align (e.g., should "about" correspond to "comment" or "description"?). As more applications adopt flexible data representations in the future, we hope better mappings will become possible.

In order to effectively maintain the illusion of a single unified data model across the user's PIM applications, Jourknow stores explicit bidirectional pointers between each subtext item and all its exported shadows. Some external APIs such as GData [1] already generate unique IDs for identifying elements; in this case, these IDs are used as-is; in other cases, Jourknow generates an identifier and stores this both with the subtext item and in a miscellaneous field of the shadow. Once this correspondence is established, maintaining synchronization is simple; Jourknow periodically polls applications using its data transfer API, and examines all items that are tagged with a subtext identifier. For each such item, it casts it back to the Jourknow representation, and compares field values; if values have changed, this indicates that the user has edited the subtext externally. Likewise, when the user updates the subtext via the codex, Jourknow first determines whether the subtext already has exported shadows; and if so, updates these shadows with new values. If a conflict is detected (which should be very rarely the case, as Jourknow propagates updates externally immediately after they occur) this conflict is indicated as a correction in the codex. Jourknow currently exports shadows to Google Calendar via the GData API, and Apple's iCal and Address Book via Applescript.

Context

The context KB consists of a chronology of observations made of the user's desktop state and actions, and of their situation/environment, which is created and maintained by PLUM. [34] PLUM executes a sequence of observer knowledge sources at a regular frequency (usually 2-3Hz), which call various facilities in the underlying operating system to yield observations. Examples of activities currently observed by PLUM knowledge sources include the identity of the application that has focus at each moment, the names and locations of any documents or web pages being viewed or edited, the user's activity in chats, writing emails, or music listening, as well as periodic desktop screen captures and the user's activity/idle state. Examples of environment/situational state captured by observers include the user's location (as perceived through Placelab [20]) and web cam snapshots of the user.. Each observation each have an associated "validity" time interval, which represents the largest contiguous time interval for which, according to the observer, the observed phenomena remained unchanged. All observations made by knowledge sources are encoded as RDF graph structures (for representational flexibility), grounded in the PLUM ontology.

Episodes and saliency heuristics

In order to find the set of episodes associated with particular text in the codex, Jourknow finds all the episodes whose intervals intersect the creation times of the text. Once associations are made between text and episodes, Jourknow extracts observations relevant to each episode by finding all observations that overlap with each relevant episode's interval. However, there may be a great number of observations; in order to prevent inundating the user with a record of their activities, we have designed *saliency heuristics* to select observations that are likely to be memorable and relevant to the user. These heuristics include, for each type of context observation, most recently observed, most frequently observed, and longest total duration. We additionally defined an "outlier" saliency heuristic inspired by TF-

IDF [30] which weights context proportionally to its total observed duration during a particular episode, and inversely proportionally to its total frequency observed across episodes. This latter heuristic has proven useful in filtering out routine visits to commonly revisited web sites, as well as intermediary pages that consumed little face time.

Structure extraction from text

To support the various modes of input described in the Interaction section, Jourknow provides three types of textual pattern analysis: simple syntactic forms (i.e., regular expressions), recursive-decent parsing, and Notation3 interpretation. As the recognition process can be compute-intensive (particularly for the recursive-decent parsing), Jourknow runs recognizers only on regions containing changed text, and executes recognizers asynchronously and independently on their own threads.

The syntactic recognizer uses a standard regular expression engine to find syntactically structured elements in the text, including tags, file paths, and URLs. The Jourknow pidgin language processor (PLP) expands the scope of Jourknow's extraction capabilities to context-free languages. We have thus far designed pidgin grammars to handle the most common found types of PIM data: events, contacts, and todos. To implement PLP, we extended NLTK_lite's *rdparser*, [2] a simple deterministic top-down parser for context-free grammars. Our modifications involved allowing the inclusion of regular expression as terminals, which we defined to match if and only if the regular expression matched the entire token. This modification greatly simplified the recognition of tokens by syntactic form (such as dates) without a separate lexical analysis phase, and enabled us to add "wildcards" to the grammars, to stand in for words or combinations of words that could not be known *a priori*. This occurred frequently in our pidgin grammars, such as with names of people, locations, or the topics of a meeting.

While enabling wildcard terminals greatly enhanced the expressiveness of our grammar language, it also dramatically increased parse ambiguity. Modifying the grammar to not requiring phrase headwords (usually prepositions such as "at" or "with") made the number of ambiguous parses combinatorially larger, but also dramatically improvd usability of the grammars, as we observed [5] that prepositions were often omitted in people's information scraps. Fortunately, we were able to tackle the complexity of parse ambiguity resolution in three ways. First, interleaving wildcard token matching into parsing made it possible to "wrap" lexical ambiguity problems into parse ambiguity resolution, greatly simplifying handling from both an implementation and user-experience standpoint. Second, because the rdparser automatically returned all possible parses for a particular sentence under the grammar, it reduced the problem of ambiguity resolution to choosing the correct parse tree from the returned set of possible parses. Finally, we devised a simple heuristic that worked well in most cases: to choose the parse tree that was broadest at its base. This corresponded to the parse tree that recognized the greatest number of separate clauses, and attached them closest to the root. This eliminated the most common source of incorrect parses, the consumption of clauses by wildcard terminals, as evidenced by errors such as interpreting the meeting pidgin expression "meeting with Michael at Stata" as a meeting **with** a person named "Michael at Stata" rather than a meeting **with** a person named "Michael" **at** a location named "Stata". Jourknow allows the user to easily override the heuristic's choice through manual selection of the correct tree.

Associating text with subtext: Timeprints

When a pattern extractor first identifies a previously unseen structure in the text, it generates a new subtext entity to represent the item. To allow future edits of the original text to properly update the correct subtext entities, it is necessary for Jourknow to be able to uniquely identify a text's corresponding subtext. To effectively and reliably support this lookup, we tag each new extracted subtext with a "time fingerprint", computed by taking the set of creation times of the source text. Jourknow maintains a hash of such fingerprints to corresponding subtext entities, and when the source text changes, the corresponding subtext's timeprint is updated to reflect the change. Unique mentions of the same text can therefore correspond to different subtext entities, and will appear as separate "shadows" (each linked to its respective note) in legacy PIM applications as described earlier.

INITIAL USER FEEDBACK

We demonstrated Jourknow to five people to solicit initial informal reactions to Jourknow's design and features. With this demonstration, we sought only to get an overview of whether people could easily understand how the various features of Jourknow related to one another, and second, to gain an impression for the features they thought were useful and/or desirable towards their own information organizational practices. As discussed in the next section, we have plans to up this work with a formal study of the system, both to measure the suitability of specific aspects (such as pidgin language recognition) as well as a longitudinal usage study to evaluate the system as a whole.

Demonstration participants were given a tour of the interface and its features, and invited to interact with the prototype for 20 minutes. Overall reactions were very positive, and all expressed enthusiasm for the text-input interface, with which, as computer scientists, they seemed immediately comfortable. They also seemed to be universally enthusiastic about the capture and association of context capture with notes. Opinions were split about whether the tagging functionality would be useful for retrieval, as opposed to using only keyword search. Three participants expressed a desire to be able to incorporate other media in Jourknow's notes, such as pictures, music and documents. One participant was impressed by the ability to edit Calendar entries bi-directionally (i.e., either by editing their note, or their appointment in their Google calendar). Another expressed a desire to be able to automatically publish notes in Jourknow to his blog.

DISCUSSION

Our work is situated between two extremes of textual information input. On the one end, we have the natural language used by everyone to communicate, with all its flexibility, inefficiency, and ambiguity. On the other end, we have the rigid, concise, unambiguous programming languages and data syntaxes used by programmers to communicate information to machines. While folklore tells us that most people cannot program and do not want to, that only rules out an extreme of the spectrum. There is lots of room in the middle. Individuals already seek conciseness (and precision) in the notes they jot to themselves. We hypothesize that current input mechanisms, such as forms, avoid data entry ambiguity by placing a significant navigational burden (and significant input limitations) upon the user. Essentially, the computer is passing the buck to the user to parse the input for them. We speculate that requiring users to obey the strictures of a relatively natural pidgin may be less of a burden. We believe that a restricted, pidgin variant of natural language can be simultaneously natural enough for people to adopt and unambiguous enough for a computer to understand, particularly since we allow input text to "detour" into arbitrary unstructured language at need. While most people may not be able to become as precise in their input as programmers, people's current use of forms demonstrates that they have the incentive and capability to accept limited constraints on how they express information so that a computer can understand it.

A potential criticism of our approach is that it is simply "NLP-lite"---that as soon as we solve the problem of recognizing arbitrary natural language, our approach will not be needed. But there are several reasons to pursue our approach. First, a complete NLP solution remains distant at this time, and is more than is needed for the simple data capture we are supporting. Thus, our approach offers many of the benefits of NLP input, sooner. Second, we note that individuals' jotted notes are generally not in natural language. They include abbreviations, ungrammatical constructions, and a variety of other language hacks to make entry more efficient. Users do not want to waste time crafting grammatically complete sentences to record information fragments. While NLP might ultimately be able to handle this unnatural language as well, note that shorthand is highly individualistic, requiring a solution that learns for each user differently---an even more challenging problem than standard NLP. Third, our approach emphasizes the value of bridging from a more naturalistic input framework to a traditional GUI output environment, in contrast to many natural language systems that assume natural language is the right modality for both directions.

Our dispatch of captured subtext to existing applications reflects a compromise strategy. While these applications provide good domain-specific interactions, one of our major arguments in favor of text is that it saves the user from having to choose a domain for the information they enter, and instead create domain-crossing collections of information that they feel are connected. It is unfortunate, then, that at retrieval time the user must again make domain-choices based on application boundaries. It would be better for the user to be able to create rich GUI visualizations that cross domains to aggregate any objects users consider connected and exploit their structure to display them well. This goal has been pursued in the Haystack system [19] among others. We believe it will combine fruitfully with the lightweight input mechanisms described here.

CONCLUSION/FUTURE WORK

The information management benefits offered by rich graphical user interfaces over richly structured data are entirely lost if the cost of entering information into those applications deters people from doing so. In this paper, we have argued that we may not have to choose between lightweight input and sophisticated retrieval and output. Text provides an excellent lightweight input mechanism that can express arbitrary information but can also express rich structure through individualistic shorthand; that structured information and related contextual information can be navigated and retrieved through rich interfaces while providing breadcrumbs to help a user locate the less structured parts. We have outlined the design of a system that minimizes the work of text entry, and uses a variety of techniques to capture subtext (structure in the entered information) and context (situational metadata about the entered information) for use in retrieval and presentation. We have also described the implementation of a system that meets the design. Our plan now is to use the affordances of Jourknow as an evaluation platform to study which features work in which contexts to best support lightweight capture of structured data.

REFERENCES

1. Google Data API.
2. Natural Language Toolkit.
3. Bellotti, V., Dalal, B., Good, N., Flynn, P., Bobrow, D.G. and Ducheneaut, N., What a to-do: studies of task management towards the design of a personal task list manager. in *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, (Vienna, Austria, 2004), ACM Press, 735-742.
4. Berners-Lee, T. Notation3 (N3), A Readable RDF Syntax.
5. Bernstein, M., Kleek, M.V., Karger, D. and schraefel, m. Information Scraps: How and Why Information Eludes our Personal Information Management Tools. *In submission to ACM Transactions on Information Systems*.
6. Blandford, A.E. and Green, T.R.G. Group and Individual Time Management Tools: What You Get is Not What You Need. *Personal Ubiquitous Comput.*, *5* (4). 213-230.
7. Connolly, D. RDF Calendar - an application of the Resource Description Framework to iCalendar Data, 2005.
8. Czerwinski, M., Horvitz, E. and Wilhite, S. A diary study of task switching and interruptions *Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM Press, Vienna, Austria, 2004.

9. Dumais, S., Cutrell, E., Cadiz, J.J., Jancke, G., Sarin, R. and Robbins, D.C. Stuff I've seen: a system for personal information retrieval and re-use *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, ACM Press, Toronto, Canada, 2003.
10. Gemmell, J., Lueder, R. and Bell, G. The MyLifeBits lifetime store *Proceedings of the 2003 ACM SIGMM workshop on Experiential telepresence*, ACM Press, Berkeley, California, 2003.
11. Harrison, B.L., Cozzi, A. and Moran, T.P. Roles and relationships for unified activity management *Proceedings of the 2005 international ACM SIGGROUP conference on Supporting group work*, ACM Press, Sanibel Island, Florida, USA, 2005.
12. Hearst, M.A. Clustering versus faceted categories for information exploration. *Commun. ACM*, *49* (4). 59-61.
13. Herman, I. and Swick, R. Resource Description Framework, 2007.
14. Hudson, J.M., Christensen, J., Kellogg, W.A. and Erickson, T. "I'd be overwhelmed, but it's just one more thing to do": availability and interruption in research management *Proceedings of the SIGCHI conference on Human factors in computing systems: Changing our world, changing ourselves*, ACM Press, Minneapolis, Minnesota, USA, 2002.
15. Iannella, R. Representing vCard Objects in RDF/XML.
16. Jambor, M., Hruby, T., Taus, J., Krchak, K. and Holub, V. Implementation of a Linux log-structured file system with a garbage collector. *SIGOPS Oper. Syst. Rev.*, *41* (1). 24-32.
17. Kalnikaite, V. and Whittaker, S., Software or Wetware? Discovery When and Why People Use Prosthetic Memory. in *CHI*, (San Jose, CA, 2007), ACM Press.
18. Karger, D.R. and Jones, W. Data unification in personal information management. *Commun. ACM*, *49* (1). 77-82.
19. Karger, D.R. and Quan, D. Haystack: a user interface for creating, browsing, and organizing arbitrary semistructured information *CHI '04 extended abstracts on Human factors in computing systems*, ACM Press, Vienna, Austria, 2004.
20. LaMarca, A., Chawathe, Y., Consolvo, S., Hightower, J., Smith, I., Scott, J., Sohn, T., Howard, J., Hughes, J., Potter, F., Tabert, J., Powledge, P., Borriello, G. and Schilit., B., Place Lab: Device Positioning Using Radio Beacons in the Wild. in *Pervasive*, (Munich, Germany, 2005).
21. Lamming, M., Brown, P., Carter, K., Eldridge, M., Flynn, M., Louie, G., Robinson, P. and Sellen, A. The design of a human memory prosthesis. *The Computer Journal*, *37* (3). 153-163.
22. Lansdale, M. and Edmonds, E. Using memory for events in the design of personal filing systems. *Int. J. Man-Mach. Stud.*, *36* (1). 97-126.
23. Lanzenberger, M. and Sampson, J., AlViz - A Tool for Visual Ontology Alignment. in *International Conference on Information Visualisation (IV'06)*, (2006).
24. Mark, G., Gonzalez, V.M. and Harris, J. No task left behind?: examining the nature of fragmented work *Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM Press, Portland, Oregon, USA, 2005.
25. McCallum, A. Information Extraction: Distilling Structure from Unstructured Text *ACM Queue*.
26. Moran, T.P., Cozzi, A. and Farrell, S.P. Unified activity management: supporting people in e-business. *Commun. ACM*, *48* (12). 67-70.
27. Rhodes, B. and Crabtree, I.B. Wearable Computing and the Remembrance Agent. *BT Technology Journal*, *16* (3). 118-124.
28. Ringel, M., Cutrell, E., Dumais, S. and Horvitz, E. Milestones in Time: The Value of Landmarks in Retrieving Information From Personal Stores *INTERACT 2003*, ACM Press, 2003.
29. Ross, L. and Nisbett, R. *The Person and the Situation: Perspectives of Social Psychology*. Temple University Press, 1991.
30. Salton, G. and Buckley, C. Term-weighting approaches in automatic text retrieval. *Inf. Process. Manage.*, *24* (5). 513-523.
31. schraefel, m.c., Smith, D.A., Owens, A., Russell, A., Harris, C. and Wilson, M. The evolving mSpace platform: leveraging the semantic web on the trail of the memex *Proceedings of the sixteenth ACM conference on Hypertext and hypermedia*, ACM Press, Salzburg, Austria, 2005.
32. Sebba, M. *Contact Languages: Pidgins and Creoles*. Macmillian, 1997.
33. Sellen, A., Fogg, A., Aitken, M., Hodges, S., Rother, C. and Wood, K., Do Life-Logging Technologies Support Memory for the Past? An Experimental Study Using SenseCam. in *CHI*, (San Jose).
34. Van Kleek, M. and Shrobe, H., A Practical Activity Capture Framework for Personal, Lifetime User Modeling. in *User Modeling*, (Corfu, Greece, 2007).