From: AAAI Technical Report SS-03-01. Compilation copyright © 2003, AAAI (www.aaai.org). All rights reserved.

Agentized, Contextualized Filters for Information Management

David A. Evans, Gregory Grefenstette, Yan Qu, James G. Shanahan, Victor M. Sheftel

Clairvoyance Corporation
5001 Baum Boulevard, Suite 700, Pittsburgh, PA 15213-1854, USA
{dae, grefen, y.qu, jimi, v.sheftel}@clairvoyancecorp.com

Abstract

Every time a user engaged in work reads or writes, the user spontaneously generates new information needs: to understand the text he or she is reading or to supply more substance to the arguments he or she is creating. Simultaneously, each Information Object (IO) (i.e., word, entity, term, concept, phrase, proposition, sentence, paragraph, section, document, collection, etc.) encountered or produced creates context for the other IOs in the same discourse. We present a conceptual model of Agentized, Contextualized Filters (ACFs)—agents that identify an appropriate context for an information object and then actively fetch and filter relevant information concerning the information object in other information sources the user has access to. We illustrate the use of ACFs in a prototype knowledge management system called *ViviDocs*.

Information Management

Developing technology for information management (IM) is a challenge because our systems cannot be based on the perfection of any single function—such as superior information retrieval, for example—but rather must derive their usefulness from an interaction of many functions. Effective IM will depend on the integration (and exploitation) of models of (1) the user, (2) the context, and (3) the application (or information purpose) with (4) the processing of source data. Integration will be the dominant factor in making information management systems useful. To aid such integration, we seek to mobilize information in the user's environment.

IM tasks are highly contextualized, highly linked to other tasks and related information—never tasks in isolation. Every time a user engaged in work reads or writes, the user spontaneously generates new information needs: to understand the text he or she is reading or to supply more substance to the arguments he or she is creating. Simultaneously, each Information Object (IO)—word, entity, term, concept, phrase, proposition, sentence, paragraph, section, document, collection, etc.—encountered or produced creates context for the other IOs in the same discourse. An effective IM system will automatically link varieties of such IOs, dynamically preparing answers to implicit information needs.

Copyright © 2003, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

To this end, rather than focus on a system that performs a single "end-to-end" function—processing a request for information or finding "similar" documents or even "answering a question"—we have been focusing on the critical components of a system (which we call "ViviDocs") that operates *behind* more ordinary user tasks, such as reading messages or writing reports. These tasks are not, explicitly, directed at finding information. But when performed in the workplace, these tasks continually generate new information needs; and to address these, we require a system that can ground a document in a structured web of authoritative information.

Agentized, Contextualized Filters

In ViviDocs, while a person reads or writes a text (an e-mail message; a report), the components of the text are continually analyzed into candidate IOs. A variety of agents are generated for each new IO. These agents identify an appropriate (typically local) context for the IO—represented by other text or information in the user's environment—and then actively fetch and filter relevant information concerning the IO in information sources the user has access to. We call such agents "Agentized, Contextualized Filters" (ACFs). They are *agents* in the sense that they operate autonomously and asynchronously; they are triggered by some event; they use their own data; and they perform specific functions on the data; and they adjust to changing conditions, potentially learning from the user's behavior (Genesereth and Ketchpel 1994). They are *contextualized* because they are anchored to specific IOs in contexts of use.

A Conceptual Model of ACFs

We define an ACF as a function that links one information object (as anchor) with another information object (output), taking into account the context of the task and the context of the user's work environment. Formally, we define an ACF as:

$$ACF_i(P_i, R_i, S_i, \theta_i, H_i, U_i, C_i, T_i, F_i)$$

where (for time/instance i) P_i represents the feature profile of the information object, R_i, the associated knowledge resources, S_i, the target sources, θ_i, the threshold, H_i, the history lists, U_i, the utility function for the user, C_i, the

processing context, T_i, the triggering condition that activates the agent, and F_i, the response function and format. We elaborate on each of these factors below.

Profile (P_i). The Profile is a representation of the information object based on its textual content. For example, in an information retrieval system, a profile representing an IO (e.g., a document or paragraph) might consist of a list of terms with associated weights to reflect their usages in the document or with respect to a document collection.

Resource (R_i). Resource refers to language resources (e.g., stop words, grammar, lexicons, etc.), knowledge resources (e.g., abstract lexical-semantic types, taxonomies or classification schemata, semantic networks, inference rules, etc.), and statistical models (e.g., term frequency and distribution counts, language models, etc.) used for processing.

Source (S_i). Source refers to the target or available information sources, accessible to the user or to the agent, in which responses to information needs may be found. In a workgroup, this might include all the user's files and the accessible files of the members of the user's team or department. In a general business setting, this might include the contents of the company intranet, extranet, and, selectively, the internet, as well as the user's personal files.

History (H_i). History consists of lists of information objects (and perhaps "scores") that have been generated by previous actions of ACFs. For example, in information retrieval with user feedback, the initial ranked list of documents considered as relevant by the system can be regarded as the history for the next round of retrieval with additional user feedback.

Threshold (θ_i). A threshold is used to control the cut-off points in decision making. Thresholds can be absolute numbers (e.g., the top 100 documents or passages), similarity scores, or confidence scores applied to retrieved information.

Utility (U_i). Utility is used to measure and rank system outputs based on the benefits they produce for the user or on the degree to which they satisfy the user's information needs minus the associated costs. Such measures are commonly used in information filtering and are typically calculated from an explicit or implicit statement of the tolerance for "noise" (the ratio of true-positive to false-positive responses) in the output.

Context (C_i). Context provides additional information that can be associated with the profile. While this concept is inherently open-ended (and subject to overuse), we restrict it to information that can be determined operationally by the system. We distinguish at least three kinds of context: (a) global context, (b) local context, and (c) focus. In an IR-like action anchored to a specific IO (e.g., word or phrase), the global context might be the document in which the IO occurs; the local context the paragraph; the focus the sentence (essentially, the proposition expressed).

Consider, for example, the following passage in a text on the German military campaign in the Soviet Union during World War II:

> *The Battle of Stalingrad represented a major turning point for the Germany Army. The German general Paulus was out-foxed by the Russian Generals by being drawn into the city. The Russians eventually wore the Germans down, cut off their supply lines, and made retreat impossible.*

The simple IO corresponding to "Paulus" has several constraining contexts. The global context establishes Paulus as a German general in WWII. Local context relates specifically to his participation in the battle of Stalingrad. Focus involves his particular role in the event, namely, being "out-foxed" by the Russian generals. If we imagine stepping through the document and selecting each such IO (e.g., person-name reference) in sequence, we can see that the general context is stable, and does not need to be updated as we move from IO to IO; the local will change frequently, from passage to passage; and focus will vary from sentence to sentence. If the user were *writing* a text, we could imagine focus changing quite dynamically, even as the user wrote a new sentence or deleted an old one.

User profiles and work-tasks can be treated as another source of context. On projects, the current set of documents that a user is working on or has access to may supply the global context, the specific document in which the information object is found can be the local context, and the immediate vicinity of the IO can be the focus.

Trigger (T_i). Triggers activate the ACFs. The action associated with opening a document or beginning to compose a message could launch a battery of ACFs. Under a GUI, triggers can take the form of highlighting, typing, clicking, etc. For example, every time the user types a full stop, an ACF can be triggered on the most recently completed sentence. Likewise ACFs could be triggered every twenty-four hours, updating the information that they associate with the IOs they are attached to.

Function (F_i). Function specifies the relation that is to be established between the IO and other information by the ACF, including the format for extracting or presenting such information. The function might be as simple as "retrieval"—finding a rank-ordered list of documents or passages—or "answer" (a simple sentence) in response to an implicit question. But the function might also be considerably more complex, such as establishing the background facts that support the proposition that the IO asserts. Functions have associated presentation requirements or formats. Formats typically require that a set of (possibly contrastive) information be developed, such as the ranked list of responses to a query, or clusters of passages that each represents different senses of a response. More ambitious combinations of functions and formats might involve providing the user with a sense of the structure of the space of answers (via topic modeling, perhaps (Evans et al. 2002)); or the location of centers of

importance (via semantic hubs and authorities, perhaps); or of related topical "regions" (via semantic-space abstractions).

ACF Parameters

Generally, of course, parameters of an ACF interact with each other. For example, our model of the user affects utility. If the user is an analyst who already knows a great deal about a topic, then we probably want to maximize the novelty aspect of any information we link to the user's work and discount the information already in the user's background (files, past work, workgroup, etc.). On the other hand, even in the case of a user whose "normal" type is well understood, based on the user's response to information or changing assignments, we may need to update or revise the user model and other parameters frequently.

The issue of parameter interaction and calibration would seem to doom the model, especially if one considers the need to adapt to specific users over time: the "training" problem could be daunting. However, though parameters can vary quite widely in theory, we observe that, for many practical application types, the actual values of parameters may be quite limited. In short, in practical use, only a few of the parameters will vary freely and these will overwhelmingly assume only a few possible values.

As an illustration, consider one of the most general functions an ACF can perform: *association*—finding relevant related material. Note that, while this might be implemented as a simple IR task, taking the text of a document as a query and searching available external sources, the proper association of information to a document is not a trivial matter. For example, a long document, taken as a query, will typically give high rank to documents (responses) that share terms with its dominant (high-frequency/low-distribution) terms. If the external sources are large, it is likely that virtually all the top-ranked responses will be biased to the "summary" or "centroid" sense of the document. Thus, in order to insure that all the parts of the document are properly represented, an association process should formulate many separate queries from the text of the document and merge results in a fashion that insures that all parts will be represented in "high-ranking responses." An ACF that performs such a task on "start up" (when a document is opened, for example) might well follow a standard procedure to decompose the document into sequences of passages (each serving as a source of a query (P)), use default resources for term extraction (R) on each passage of approximately paragraph size, and target a default large external source (S). Such an ACF might ignore context (C) and history (H), since the document itself is term rich and the user's session is just beginning, being triggered (T) upon opening the document. The function to be performed—in this case, multi-pass IR (F)—can be specified to establish a local cache of material that will be of high value if the user wants to explore topics or answer questions that arise in reading the text. Thus, the only open questions relate to what the operational interpretation of utility (U) and threshold (θ) should be. In this regard, a variety of heuristics may prove serviceable, e.g., (1) insure that each passage brings back at least *n* documents and all documents (up to a maximum, *m*) that score above the threshold; (2) vary the threshold for each passage based solely on the scoring potential of the passage against the data being searched; (3) aim for a final cache of documents in the range of 100 to 10,000. This might be achieved by ranking the results of each passage-query using normalized scoring—dividing the term score of each responding document by the term score of the first-ranked document—using a fixed threshold, e.g., 0.7 or 0.6 normalized score, and returning (and caching) the top *n* responses and any other responses (up to the *m*th) that score at or above threshold. Since we know how big the document is (the count of the number of passages we extract from it), we can set *n* and *m* to insure that the resulting information cache is in the target range (e.g., 100 to 10,000 documents).

Figure 1 gives the parameter settings in schematic form for a **FindRelevantDocs** ACF that can effect the *association* function described above. Note that the actual implementation of an ACF such as this one requires a host of supporting operations, such as document-structure processing (e.g,, to find passages), term extraction (e.g., NLP to identify the unit features of the profile for each passage), an indexing system (for the external sources), and a filtering or IR system with mechanisms for using reference data (resources) to weight and score terms and for enforcing thresholded retrieval. In addition, these must be integrated with the system's document-handling and editing functions and GUI. However, if such supporting operations are available, the interpretation of an ACF is straightforward and the processing (e.g., multi-pass retrieval) can be made quite efficient.

FindRelevantDocs

Profile:	<terms in $Passage_i \in Document$, passage-count=*I*>
Resource:	<English lexicon, English grammar>
Source:	<specified *Source*>
History:	<empty>
Threshold:	<all documents *d* in *Source* to rank = max(*n*,min(count(norm-score(*d*)≥0.7),*m*)), where n=100/*I* and *m*=10,000/*I*>
Utility:	<not defined>
Context:	<empty>
Trigger:	<opening of *Document*>
Function:	<retrieve documents from *Source* for each $Passage_i$; cache results>

Figure 1: Schematic FindRelevantDocs ACF

The essential observation we make is that the number and type of parameters in an ACF, itself, is not a barrier to ACF development. In fact, we believe that the total number of ACF types required in order to establish full and rich functionality in a system such as ViviDocs probably is

less than fifty and possibly less than twenty five. Most of these will have a small number of variable parameters in practice, related directly to the type of function (e.g., retrieval vs. question-answering) the ACF performs.

Types of Information Needs and ACFs

The user's information needs, whether implicit or explicit, can be organized in a hierarchy of increasing complexity. On the first level, we have implicit information needs that are local to the information objects mentioned: factoids (such as those supplied by current QA systems), definitions, localizations, elaborations on information objects mentioned. On a higher level, we have argumentative and discovery needs: authoritative evidence for facts, recognition of arguments being made, finding support for and against arguments, discovery of unmentioned information (e.g., third parties associated with mentioned parties).

Corresponding to the types of information needs, we design ACFs that generate a hierarchy of *investigative discourse* answer types. These answers range from the relatively simple to the very complex and include (a) *definitions* ("factoids" such as *who*, *what*, *when*, *where*, etc.), (b) *descriptions* (contextualized facts), (c) *elaborations* (information that expands the background of a contextualized fact), (d) *explanations* (a set or sequence of facts that are causatively related to one another or the anchor IO), (e) *arguments* (a set of facts that reflects alternative points of view on the anchor IO), (f) *synthesis* (a set of facts ordered to reflect steps in a logical process, oriented to a goal or outcome), and (g) *discovery* (a set of facts represnting new knowledge).

The simpler types of information needs, such as *definitions*, *descriptions*, and *elaborations*, may be addressed with functions such as small-passage-level IR or question answering, especially if these can be targeted to sources that are designed to provide answers—dictionaries; encyclopaedias; gazetteers; phone and address books; company directories; FAQ databases; etc. Even over free texts, we can design processes that will retrieve a large amount of information, cluster it (for organization), and then order related information for complementary coverage of a topic.

Clearly, some types of information needs may be very difficult to satisfy (even if a human agent were addressing them). In increasing order of difficulty, *explanation*, *argumentation*, *synthesis*, and *discovery* are at the core of higher intelligence. We do not imagine that there is a facile solution to the challenges they pose. However, we do believe that selective components of such functions can be automated and will be useful even though they may be primitive. For example, the explanation of an event or conclusion may lie in antecedent information. The set of such prior information, assembled, sorted for topic, and chronologically presented to the user, may be precisely the response required to support the user's own, efficient discovery of an underlying cause.

We believe that it is less important that an ACF perform a specific function flawlessly than that an ACF perform a function well enough to provide the user with information that the user can use to complete the function efficiently.

Networks of Information

When ACFs are activated, they produce a network of linked IOs, with the following features.

- **Asymmetric** The ACFs serve as links that process the given information object and pass information from it to another information object. For example, in ViviDocs, a **FindRelevantDocs** filter starts with a query and returns a list of ranked documents that are relevant to the query. A **FindDescriptionWhere** filter starts with a question and returns a list of documents with location names. In general, the linking between two information objects is *directional* from the anchor to the output.

- **Dynamic** Links are created virtually between information objects that may themselves be in flux. The relation of one object to another—which might serve as a basis for establishing context, for example—can change as a result of information being passed.

- **Personalized** The interpretation and processing of information objects at linking time reflect the user's unique perspectives. For example, consider the information request "find documents about ATM." In the global context of a financial analyst, the appropriate responses are likely to be related to Automated Teller Machines (ATMs), while in the global context of a network engineer, the appropriate responses are likely to be related to Asynchronous Transfer Mode (ATM).

- **Contextualized** The interpretation and processing of information objects at linking time depends upon context scope. In the Battle-of-Stalingrad example, the information returned about Paulus in the local context is different from the information about Paulus in the global context, which tells us about the person and his career.

- **Structured** The information that is found by ACFs naturally lends itself to a structured interpretation. For example, different ACFs (anchored to different IOs in a user's document) may "touch" the same passages in external sources or in the local store of information associated with the document many times. Any such individual passage is thus "validated" as useful to the document by many independent agents; it can be interpreted as an "authority" passage for the document. Similarly, if an external document is the source of many separate passages, each of which is referenced by independent ACFs, that document can be regarded as playing the role of a "hub" document. In short, the links established by ACFs in the set of related documents and passages create a quantifiable, network structure, directly anchored to the user's task.

The notion of linked information was already present in the original MEMEX vision (Bush 1945). Many people regard the World Wide Web as the practical realization of

MEMEX since the Web offers a concrete example of linked IOs. Parts of a document may be linked to whole other documents or parts of other documents; the link lattice can be used to move from point to point along pathways of relevance (or, at least, association). But the network itself is relatively static and the types of links are quite general—and must be created "by hand," explicitly. Thus the possible interpretations of information must be decided at link time—by individuals creating links, reflecting their unique perspectives. The possibility that the "same" information might be linked to multiple, distinct other objects, depending on the information needs of a given user, cannot be accommodated. Such a static approach is limited. True "knowledge networks" will be subject to constant change and "re-linking" of information, dynamically. Thus, the original vision of MEMEX—as a knowledge network—has not been realized in the Web.

Illustration and Use Case

We have implemented a prototype to study the behavior of ACFs. The prototype only demonstrates a limited set of the design features of ViviDocs. For instance, in the current version, history lists produced at different times are not maintained; only immediate history lists are available. Also, there is no modeling of contexts at different times; only the latest contexts are maintained. Utility has not been incorporated (except in default settings).

ViviDocs is build on the back of the CLARIT information-management system (Evans et al. 1991; Evans and Lefferts 1995), which encompasses numerous IM functions ranging over NLP, extraction (of typed entities), IR, filtering, question answering, and concept clustering. In contrast, the current GUI supports little more than reading and writing a text and is not integrated with other productivity software, such as e-mail. We present examples below.

An Example Based on Writing

When the user begins to write a text, ViviDocs attempts to anticipate the types of information the user may need. Figure 2 shows the simple ViviDocs screen editor, in which the user has just typed "Hostage taking has become a contemporary crisis." The period at the end of the sentence is a trigger (T) that activates several ACFs working in the background. Here the IO is by default the new text "hostage taking has become a contemporary crisis." The profile (P) for this IO is represented as a vector of terms that have been extracted using CLARIT NLP, which uses lexicons and grammars to identify linguistically meaningful units (R) from text and also uses a reference database (R) to obtain occurrence (distribution) statistics. The following list shows the terms and their distribution statistics:

contemporary crisis: 0
hostage taking: 22
hostage: 587
contemporary: 2387
crisis: 4149
taking: 12042

The **FindRelevantDocs** ACF uses this information to create a query over an available source (S), a collection of AP newswire articles. The threshold (θ) is set to retrieve the top 100 relevant documents. The response is cached as new IOs (F). Both the history list and the context are initially empty.

Other ACFs begin to work on the cached IOs as soon as they are available. Each of these ACFs performs a specified function, using the IOs in the text as anchors. If the user wants to see different factual aspects of the topics that have been fetched in the background, he right-clicks the mouse and gets a menu of the set of ACFs that have been activated (Figure 3). Selecting the **Description→Where** menu item displays the responses produced by the **FindDescriptionWhere** filter (Figure 4). The **FindDescriptionWhere** filter reformulates the original written text as a question, and produces documents relevant to the question by specifically finding information related to locations. The additional resources (R) it exploits include resources for extracting locative entities. Now, the history list (H) contains the ranked documents returned by the **FindRelevantDocs** filter, which serves as local context for the locations.

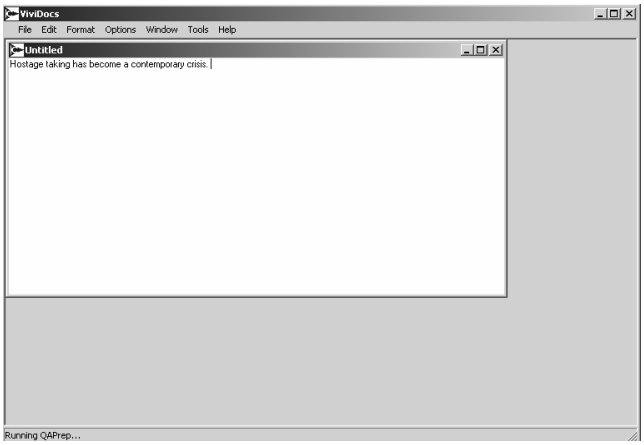

Figure 2: The editor screen of ViviDocs

Figure 3: Menu for specifying results from ACFs

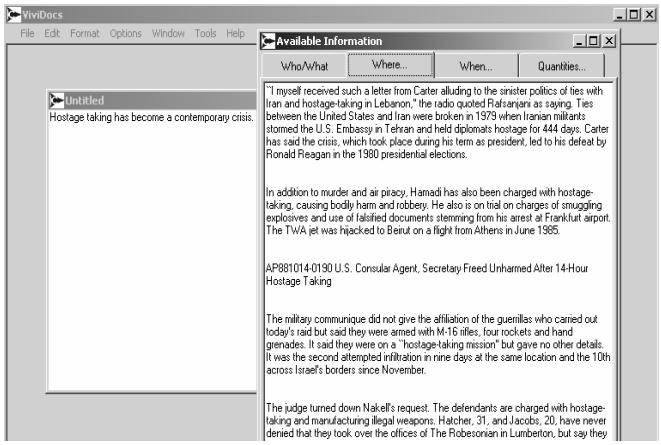

Figure 4: Responses of the FindDescriptionWhere filter

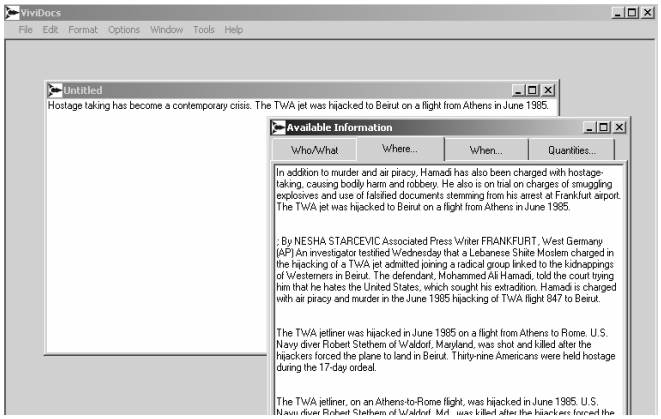

Figure 5: Updated responses of FindDescriptionWhere

After the user browses through passages on hostage taking in different locations, he wants to know more about the hijacking of the TWA jet from Athens to Beirut in June 1985. So he cuts the text "The TWA jet was hijacked to Beirut on a flight from Athens in June 1985" from the results form and pastes it to the original editor. Highlighting (T) of the new text in addition to the original text updates the linking maintained by the ACFs. Now selecting the **Description→Where** menu item returns passages that discuss the hijacking of the TWA jet specifically (Figure 5).

The parameters used in the two ACFs discussed above are given in Figures 6 and 7. Note that information passages (IO_2) created by the first ACF (**FindRelevantDocs**) are in the history list and serve as the appropriate task context for future use. The retrieved passages are indexed into a local database (D_2), which subsequently is the source used by the second ACF (**FindDescriptionWhere**). Upon right-clicking of the mouse and selecting of the **Description→Where** option in the GUI, the **FindDescription-Where** agent is activated and formulates the original IO as a *where* question to extract factual answers from the source (D_2). Currently, instead of returning the exact factual answers, the agent brings back passages that potentially contain the correct answers.

FindRelevantDocs

Profile:	<contemporary crisis: 0; hostage taking: 22; hostage: 587; contemporary: 2387; crisis: 4149; taking: 12042>
Resource:	<English lexicon, English grammar >
Source:	<indexed AP88 database built with 3-sentence passages>
History:	<empty>
Threshold:	<N=100>
Utility:	<not defined>
Context:	<empty>
Trigger:	<typing of ".">
Function:	<retrieval ; caching (=IO_2)>

Figure 6: Instantiated FindRelevantDocs ACF

FindDescriptionWhere

Profile:	<contemporary crisis: 0; hostage taking: 22; hostage: 587; contemporary: 2387; crisis: 4149; taking: 12042>
Resource:	<English lexicon, English grammar >
Source:	<indexed database built based on IO_2>
History:	<IO_2>
Threshold:	<N=10>
Utility:	<not defined>
Context:	<IO_2>
Trigger:	<mouse click and menu selection>
Function:	<answer-where>

Figure 7: Instantiated FindDescriptionWhere ACF

An Example Based on Reading

When a user begins to read a document in the current version of ViviDocs, the system segments the document into passages (paragraphs) and the **FindRelevantDocs** ACF polls external sources for information that is related to the document, as described above. The returned passages/documents constitute an information repository that can subsequently be used by other ACFs to find more detailed information. These other ACFs proceed through the document, passage by passage, and attempt to perform their respective functions for each IO they encounter. In such cases, the local context will be the passage itself and the focus will be the sentence or proposition in which the IO is located. At any time, if the user selects an IO or a local context, the system is prepared to return the information that has been found by the ACFs that operated on that IO. Typically, this results in sets of information that reflect multiple perspectives on the IO.

In the case illustrated in Figure 8, the user has opened an AP-newswire document on Bush's presidential campaign (in 1988). The article notes that the Iran-Contra affair and the associated indictments could be a liability for Bush. If the user wants to know more about who was involved in the Iran-Contra scandal, the user can activate the **Description→Who** filter, which brings back passages with the relevant entities highlighted, as shown in Figure 9. For

this ACF, the highlighted entities include person names and organization names.

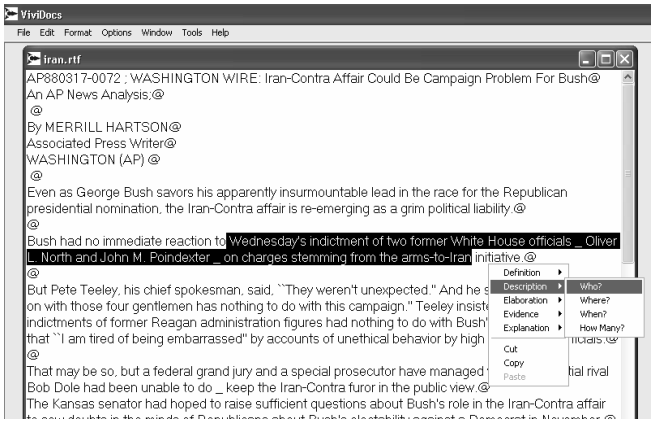

Figure 8: Document opened for reading in ViviDocs

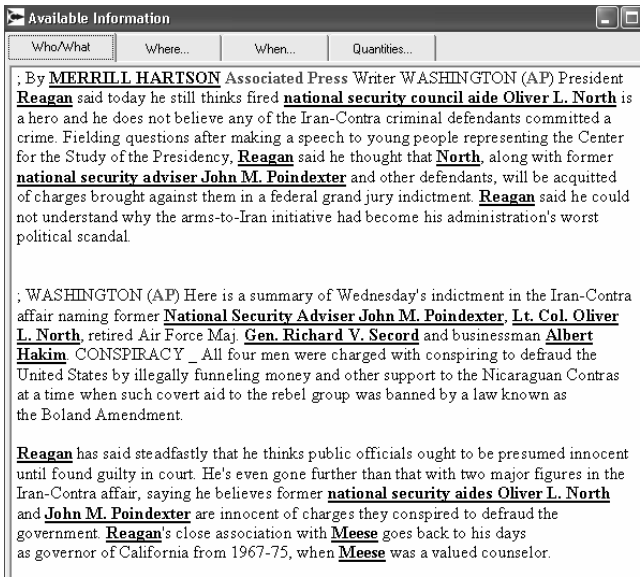

Figure 9: ACF responses relating to "Who"

Note on Details of Functionality

To summarize, in our current implementation, for both the reading and writing tasks in Vividocs, the ACFs are based heavily on two IM functions: retrieval/filtering and question answering. The retrieval/filtering ACFs use information objects (e.g., a sentence, a passage, or a whole document) to bring back associated passages from user-selected databases. The returned passages together serve as an information repository and context for a battery of other ACFs that establish relationships (such as definitions, description, evidence) between information objects in the user's document and the external sources.

The ACFs that establish the *description* relationships rely on a question answering system that utilizes typed entity extraction and passage re-ranking. The QA system first retrieves small-sized passages (e.g., 3-sentence passages in our demo) that potentially contain the factual answers that are of interest to the user. These passages are then re-ranked taking into account the extracted entities associated with the user's interests (the selected aspect) and the retrieval scores. For example, if the user is interested in the *who* aspect of a particular topic, the **FindDescriptionWho** filter will rank higher the relevant passages with person and organization names. The extracted entity types in the current system include person names, organization/office names, country names, place names, time, currency, and numerical values.

Challenges for Research

Various attempts at developing IM systems such as ViviDocs have been proposed and attempted over the past decade. In general, the central themes of such work have involved the problems of (1) managing or exploiting context or (2) anticipating user's needs.

With regard to capturing context, much work has focused on improving context for single queries, either explicitly or implicitly. People often make context explicit, as when they type additional terms to help disambiguate an information need. For example, if a user is looking for a personal homepage on the web, he or she could contextualize or constrain the query by adding the word "homepage" to the name of the person in the query. This will substantially improve the relevance of the information retrieved. Web search engine such as Google.com are increasingly relying on linguistic techniques, such as entity extraction, to provide more context for short queries.

Another attempt to capture context has been the development of niche browsers that focus on providing specific types of information such as research reports or stock prices. An example of such a browser is provided by ResearchIndex.com whose inherent implicit context (target domain) is research papers. Other examples include FligDog.com (for jobs) and HPSearch.com (for computer scientists).

A number of document-centric approaches to capturing context have been proposed in the literature. Generally, most approaches try to capture context from the documents that are currently being viewed or edited by the user. One such system is the Watson system (Budzik and Hammond 2000). Watson attempts to model the context of user information needs based on the content of documents being edited in Microsoft Word or viewed in Internet Explorer. The documents that users are editing or browsing are analyzed by a heuristic term-weighting algorithm, which aims to identify words that are indicative of the content of the documents. Information such as font size is also used to weight words. If a user enters an explicit query, Watson modifies the query based on the content of the user's current document and forwards the modified query to web search engines, thus automatically adding context information to the web search. Thus, in the Watson system, though the user is required to compose the query, the system derives constraining context automatically.

Watson's mode of operation is similar to the Remembrance Agent (Rhodes and Stamer 1996; Rhodes

and Maes 2000), which indexes specified files, such as email messages and research papers, and continually searches for related documents while a user edits a document in the Emacs editor.

Recently, a number of new approaches to IM have been proposed based upon anticipating the information needs of users. The Document Souls System (Shanahan and Grefenstette 2001) is designed to annotate documents actively with various types of related information that is available on the internet or an intranet. Document Souls specifically tries to anticipate the information needs of a user. When a document is opened, it is associated with a "personality" (i.e., a collection of information services and lexicons). This personality then identifies information objects in the current document, which are subsequently annotated with links to other related information that may help the user. The text of the information object, the surrounding context, along with global information such as the topic of the document or the surrounding subdocument is used to construct queries that are submitted to various information sources (e.g., databases; folders; automatically selected regions of the classification schema of an internet search engine; etc.). This process of annotation is performed periodically.

Another example of an anticipatory system is Autonomy's Kenjin program (www.kenjin.com). Based on the documents a user is reading or editing, Kenjin automatically suggests additional content it derives from from the web or local files.

ViviDocs clearly follows in the tradition of such past efforts at extending the relevance and functionality of IM systems. However, ViviDocs attempts to generalize the model of relations that a document can have to external information sources and implements a number of specific functions, such as question answering and adaptive filtering, that go beyond simple information retrieval. ACFs are explicitly designed both to promote multifaceted associations among information objects and also to facilitate the interaction of filers based on feedback and modifications of context.

Though the current set of ACFs is limited, the ViviDocs system demonstrates novel functionality. Even in the modest prototype, one can see surprising effects. Our future work will focus on extending the number and variety of ACFs, completing the integration of advanced IM processing into the system, and refining the model of the user, the work group, and the network of linked information generated via ACF actions. Our challenge is to bring the system to operational completion and to begin experiments to test the hypothesis that ACFs can make tasks more productive and efficient and can support users in the most creative elements of their work—discovery and integration of new knowledge.

Acknowledgements

We thank three anonymous reviewers for their comments on an earlier version of this paper and for their constructive suggestions for improvements. The authors remain solely responsible for any deficiencies in the work and in the interpretation of the good advice of others.

References

Budzik, J., and Hammond, K.J. 2000. User interactions with everyday applications as context for just-in-time information access. In *Proceedings of the 2000 International Conference on Intelligent User Interfaces*, New Orleans, Louisiana: ACM Press.

Bush, V. 1945. As we may think. *Atlantic Monthly*, 176(1) (Jul):101−108.

Evans, D.A.; Ginther-Webster, K.; Hart, M.; Lefferts, R.G.; and Monarch, I.A. 1991. Automatic Indexing Using Selective NLP and First-Order Thesauri. In A. Lichnerowicz (Editor), *Proceedings of RIAO '91*. Amsterdam, NL: Elsevier, 624–644.

Evans, D.A.; Grefenstette, G.; Shanahan, J.G.; Sheftel, V.M.; Qu, Y.; and Hull, D.A. 2002. Modeling QA as investigative discourse: creating networks of functionally-linked information objects. *ARDA AQUAINT Workshop*, Monterey, California.

Evans, D.A., and Lefferts, R.G. 1995. CLARIT–TREC Experiments. *Information Processing and Management*, 31(3):385–395.

Evans, D.A.; Shanahan, J.G.; Xiang, T.; Roma, N.; Stoica, E.; Sheftel, V.M.; Montgomery, J.; Bennett, J.; Fujita, S.; and Grefenstette, G. 2002. Topic-specific optimization and structuring. In E.M. Voorhees and D.K. Harman (editors), *The Tenth Text REtrieval Conference (TREC-2001)*. NIST Special Publication 500-250. Washington, DC: U.S. Government Printing Office, 132–141.

Genesereth, M.R., and Ketchpel, S.P. 1994. Software agents. *Communication of the ACM*, 37(7).

Rhodes, B.J., and Maes, P. 2000. Just-in-time information retrieval agents. *IBM Systems Journal (special issue on the MIT Media Laboratory)*, 39(3&4): 685-704.

Rhodes, B.J., and Starner, T. 1996. Remembrance Agent: a continuously running automated information retrieval system. In *Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi Agent Technology*, 487–495.

Shanahan, J.G., and Grefenstette; G. 2001. Meta-document and method of managing meta-documents. European Patent # EP1143356. Pending (Filing Date: April 4, 2001.)

Zhai, C.; Jansen, P.; Stoica, E.; Grot, N.; and Evans, D.A. 1999. Threshold calibration in CLARIT adaptive filtering. In E.M. Voorhees and D.K. Harman (editors), *The Seventh Text REtrieval Conference (TREC-7)*. NIST Special Publication 500-242. Washington, DC: U.S. Government Printing Office, 149–156.