Uncovering the to-dos hidden in your in-box

D. M. Sow
J. S. Davis II
M. R. Ebling
A. Misra
L. Bergman

In this paper we present SCOUT, an application that examines the machine-generated messages within the in-box of an e-mail application, extracts from these messages information regarding the tasks the recipient is asked to perform, and displays these messages in a graphical interface where they are grouped by context. The tool is intended for business managers who receive daily a large number of machine-generated messages that require some action be taken. SCOUT uses the IBM Unstructured Information Management Architecture (UIMA) framework to apply rule-based reasoning for identification of tasks, and it uses contextual data to customize the presentation of task information to the user. SCOUT's open, extensible architecture allows the use of alternate inference models (such as machine learning algorithms) as well as the integration of additional context sources and client interfaces. SCOUT was well received by the participants in a small evaluation study.

INTRODUCTION

The beginning of the 21st century has been marked by an explosion of electronic information. We often find ourselves inundated with information that reaches us through a variety of channels, including e-mail accounts, voice-mail recorders, and most recently, text-messaging clients on our cellular phones. This amounts to a deluge of requests, notices, scheduling invitations, and the like. We owe the information explosion to the proliferation of inexpensive and readily available technology that has led to what most would agree is both a blessing and a curse. Whereas information facilitates efficient business and social networking, it also has become a major burden. Humans simply cannot accommodate massive information input.

For economic reasons, e-mail is the most acute and widespread facilitator of this explosion due to the virtually free cost of generating and distributing e-mail messages. Even when we ignore spam and focus on legitimate e-mail (from trusted and welcome senders), we find that e-mail is still a major problem.[1–3] One user reports receiving well over 70 legitimate messages per day.[4] Anecdotal evidence suggests that his experience is not unusual. In an enterprise environment, a significant portion of these legitimate e-mails are associated with tasks that must be acted upon by the recipient.

A key challenge posed by e-mail inundation is how to effectively manage the tasks and activities that are

associated with e-mail messages. Herein lies the goal of our work: to help users manage their tasks effectively. We consider a task to be a particular kind of activity. Moran defines "activity" as a set of mental or physical actions carried out by persons.[5] Through composition, activities can contain subactivities, which can themselves contain subactivities. In this vein, we define a *task* to be an atomic level activity, one that may not contain subactivities. We focus only on tasks that are communicated by e-mail messages, such that there is at most one task per message. As an example, the process of bidding for a product on eBay** is an activity containing many subactivities. One e-mail associated with this process alerts the recipient that he or she has won an auction. The task for the recipient contained in this e-mail is to initiate a payment to the seller.

In this work, we focus on the population of business managers who receive daily a large number of legitimate, machine-generated e-mails, such as the ones that are generated by business processes within a large enterprise. We present here a practical solution for dealing with such e-mail in the form of a task management tool called SCOUT,[6] which uses contextual information about the user and the environment to recognize, filter, sort, organize and execute tasks associated with e-mails. By using information from pervasive sources (i.e., ubiquitous computing devices), SCOUT alleviates some of the problems associated with e-mail overload by presenting the core information to the recipient in an efficient and well-organized fashion.

We hypothesize that tasks contained within e-mail messages can be automatically identified for presentation within SCOUT. Tasks can be contained in one of two types of e-mail messages: human-generated and machine-generated. For our purposes, the salient difference is that human-generated messages tend to be unstructured, whereas the contents of machine-generated messages have a regular structure. To simplify the problem, we focus on machine-generated messages. We assume that every machine-generated message is associated with some business process (e.g., the eBay bidding process or the expense reimbursement process in an enterprise), that we only have access to e-mail messages generated by business processes, and that other than inspecting the e-mail messages themselves, we have no knowledge of the syntactic structure used in these messages. Furthermore, we assume that we make no modifications to messages or to the business processes that generate them.

SCOUT tracks a set of registered task types, each of which corresponds to a business process. When SCOUT identifies an e-mail message associated with a business process, the task contained within that e-mail message is specified in a document by using an Extensible Markup Language (XML) dialect called TaskML. A task description contains the following attributes:

- Type—the task type represented by a label unique to a business process or transaction associated with the task (e.g., a bidding transaction on eBay, a password update at Amazon.com Web site).
- Subject—a summary description of the task (e.g., you have won the auction)
- Person—an optional list of persons associated with the task (e.g., a collaborator who can help complete a task)
- Deadline—an optional deadline by which the task must be completed
- Thread—the set of related messages associated with the activity containing this task
- Comments—free-form comments associated with the task
- Status—the state of completion of the task

By automatically identifying tasks within e-mails generated by business processes, SCOUT helps make users aware of the tasks awaiting their attention. Furthermore, by pulling these e-mail messages into a task management system, it reduces the number of legitimate e-mail messages the user must process each day.

SCOUT provides three main functions: e-mail analysis, context-based task presentation, and context-based task reminding.

1. *E-mail analysis*: An e-mail analysis engine recognizes incoming e-mails as being associated with known business processes. Such e-mails are then parsed and further analyzed to extract task information relevant to that process.
2. *Context-based task presentation*: SCOUT uses context associated with a task so that it can be presented in a graphical interface that is customized according to the viewer.

3. *Context-based task reminding*: To extend SCOUT beyond the desktop, context-based reminders enable task-related messages to be sent to users on pervasive devices. Users can specify contextual criteria to trigger the reminding process (e.g., if my task is to pick up a package, alert me when I am in the vicinity of the mail room; if my task involves Steve, alert me when we are both available).

The e-mail analysis function is implemented using Unstructured Information Management Architecture (UIMA) annotators. UIMA[7] is a component-based software framework used for the development of applications that process unstructured information. It focuses on text analysis and isolates the core algorithms that perform text analytics from system services such as storage of data, communication between components, and visualization of results. By offering a framework with well-defined application programming interfaces (APIs), UIMA allows developers to share and combine text analysis algorithms in order to build complex applications.

The rest of the paper is organized as follows. In the next section, we review related work. In the following section we introduce the SCOUT application, describe the way in which the application requirements were defined, and describe the two interfaces to SCOUT, the Web portal and the e-mail client. Next we present the context information used by SCOUT, the sources of that information, and the way in which additional context is derived. We then describe the SCOUT architecture and give an overview of the e-mail analysis components. We present results of a pilot study and conclude with some final comments, including ideas for future work.

RELATED WORK

Moran and his colleagues identified several *metatasks* required for efficient task management[2,5]:

- *Creating awareness* of the core task and related metatasks
- *Prioritization* of tasks
- *Scheduling* of task appointments
- *Completion* of task prerequisites
- *Monitoring* of task status
- *Notification/reminders* of partially completed tasks
- *Delegation* of tasks through reassignment

An important focus in task management is the awareness aspect. Although task management has received a great deal of attention in the literature,[8–11] most approaches tend to disregard the awareness problem. A notable exception is the work of Cortson-Oliver et al.,[12] which deals with general e-mails. They propose SmartMail, a prototype task-extraction system that uses linear support vector machines (machine-learning method used for classification) and linguistic rules to analyze unstructured e-mails. Their technique produces task-focused summaries of action items detected in e-mails. With such a wide scope on general e-mails, their solution has had only modest predictive success.

Another exception is the work of Bennett and Carbonell[13] describing a system that tries to identify the action items contained in unstructured e-mails. They compared a standard unigram (1st order Markov) approach to an n-gram ($n - 1$ order Markov) approach applied at both the document and sentence level. They found that n-grams applied at the sentence level are most effective, achieving accuracies of 0.8092, 0.8145 and 0.8173 for a k-nearest neighbor, naïve Bayes, and support vector machine classifier, respectively. In contrast to this work, SCOUT limits the e-mail that it considers to those items that arrive from semi-structured business processes. In the case of one SCOUT user with an e-mail corpus consisting of 2,269 messages, the observed accuracy was 0.9996; similar results were obtained for other SCOUT users.

Much of the work on automatically classifying e-mails aims at automatically placing e-mail messages into appropriate folders. Examples of work addressing the filing problem include Segal and Kephart's MailCat system,[14] and the work of Bekkerman et al. with e-mail data from Enron Corporation and SRI International.[8] Similar work has also been performed by Dredze et al.,[15] who focus on automatically classifying each incoming e-mail message according to the activity to which it belongs. In the TaskPredictor system, Shen et al.[16] extend this work by using incoming (unstructured) e-mail messages to predict a user's activities. The focus of all this research, however, is on the classification of e-mail messages and on predicting a user's activities, and not upon the identification of action items within e-mail messages. In contrast, SCOUT attempts to help users identify and manage the tasks that are contained within e-mail messages.

Another related body of work focuses on extracting information from text-based documents, such as e-mail messages and Web pages. McCallum provides a good overview of the challenges of information extraction,[17] including the trade-offs involved in the use of various techniques. He argues that rule-based approaches, such as the one used in SCOUT, only work on relatively simple text within applications of limited complexity. Business process e-mails are generally verbose, but contain relatively simple requests. Furthermore, assuming future access to the business processes that generate the e-mails, a reasonable long-term solution would not focus on information extraction from the e-mail messages, but on the use of a task mark-up language from which SCOUT entries and e-mail messages could be generated.

Tomasic and his colleagues[18] describe a virtual information officer (VIO) that accepts e-mailed requests for updates to corporate databases and returns partially filled out forms for user confirmation. Like VIO, SCOUT is interpreting an incoming e-mail message to identify the underlying task. Unlike VIO, SCOUT's focus is on helping users manage tasks presented to them by business processes. If SCOUT were used in conjunction with VIO, SCOUT would classify the partially filled out forms sent to the user for confirmation as a task from a business process. In addition, SCOUT differs from VIO in that it employs a rules-based approach that does not require extensive training to support each new business process; a new process can be supported by SCOUT with a minimal investment of time.

THE SCOUT APPLICATION

In this section, we present the design and realization of the SCOUT application. We begin by discussing the application requirements and how they were determined. We then present the two application interfaces that we have built—one within a portal environment and the other within an e-mail client.

Application requirements

There are many ways to support the previously outlined metatasks in a task management system. We collected application requirements in a user study that involved a focus group in a two-phase process. First, a group of seven participants were interviewed, and their comments were incorporated within a tentative set of requirements. The group consisted of managers (three first-level and four second-level) representing the target population for the tool. In the second phase, sketches of a proposed user interface were reviewed by five additional participants, three first-level managers and two second-level managers, and their comments were incorporated within the final set of requirements.

The participants felt strongly that the task management interface should emphasize simplicity with terse, relevant information displays. As a result, we decided to limit the set of functions to basic ones.

Another issue that our managers brought up was the choice between a Web-based user interface and a client-based user interface. In our survey population approximately half of the participants preferred a SCOUT implementation that was accessible via the Web, whereas the others preferred a SCOUT client integrated with their e-mail client, which would periodically synchronize with a back-end server, allowing them to process tasks while operating in a mobile environment.

Beyond the preceding design considerations, our survey group expressed interest in the following list of novel features for managing e-mail complexity:

1. *Automatically generated to-do list*—This was the primary feature that our user study participants requested. Of the managers we surveyed, only one actively used the to-do list feature of the available personal information manager tools. The primary hindrance to using to-do lists cited by users was the burden of manually populating and maintaining such lists.

2. *Rich views of to-dos*—Related to the issue of automatically populating the to-do lists is enabling access to rich and varied views of these lists. Such views depend on the ability to associate attributes (e.g., colleagues, task deadline, task type) with tasks. Manual assignment of these attributes is a tedious process. The ability to automatically extract e-mail-based tasks, create to-do entries for these tasks, and assign appropriate values for attributes was deemed extremely useful by our study group.

3. *Communication and collaboration support*—Machine-generated tasks often contain subtasks that require human-to-human communication or collaboration. Many participants expressed interest in

having a task management system that allows users to associate e-mails or other information to automatically generated tasks. In addition, such a task management system must allow users to delegate tasks to others.

4. *Automated scheduling*—The automating of scheduling tasks received a lukewarm response. Although no participant expressed interest in a completely automated tool able to schedule times for performing tasks without their explicit approval, there was some support for a feature that assists with task scheduling. For example, some participants thought that an informational calendar showing deadlines and proposing start dates for long-running tasks might be beneficial.

5. *Automated notification*—Participants thought that notification of imminent and urgent deadlines through channels other than e-mail would be beneficial, as long as the notifications were completely unobtrusive and well-timed (e.g., not sent during meetings). Such notification mechanisms had to take into account the users' context in order to make intelligent decisions. A few participants were adamant in opposing the idea of any notification.

With these requirements in mind, we designed two interfaces to SCOUT: a portal-based implementation and an implementation integrated into an e-mail client application. In spite of the significant additional implementation effort, we decided on a dual interface approach because of the strong recommendations from our study group.

The SCOUT Web portal application

The SCOUT Web portal application (SCOUT portal, for short) presents to the user a dashboard that lists all pending tasks and their attributes. ***Figure 1*** shows a screen capture of a typical task list in the SCOUT portal.

Upon visiting the portal, the user can view all pending tasks, arrange and prioritize them according to several automatically extracted attributes such as task due date, associated users, associated formal process, and task subject. As shown in Figure 1, in this view there is a `Sort by` pull-down menu in which the user can select different criteria for sorting the tasks. This list can be sorted by importance, type, subject, deadlines, or upcoming meetings. In the task list view shown in Figure 1, tasks are sorted by upcoming meetings. Taking a closer look at the fourth row of the task list reveals that the corresponding task is of type `EASubmission`, which means that it is an expense submission task generated by a business process whose name is shown in the type column. The subject is `Trip to WMCSA`, and its deadline is `12-16-2005`. In addition, SCOUT informs the user that it is `Maria R. Ebling` who submitted this travel expense. Should the user want to meet with Maria to discuss this task, the next meeting with Maria is scheduled to take place on `12-19-2005`, as shown in the upcoming meetings column.

By presenting all pending tasks in a single, well-organized interface, SCOUT helps make users aware of the tasks awaiting their attention; it also helps them prioritize those tasks and monitor the status of pending tasks—all important metatasks associated with efficient task management.

Task-specific details can be viewed by clicking on the task `Subject` link. ***Figure 2*** shows the task detail view when the user clicks on the fourth task of the task list shown in Figure 1. From this screen, the user can edit the importance rating of this task as well as its deadline and status. (The task status can be monitored in the task list view.) In addition, if a task needs to be delegated to a colleague, clicking on `Delegate Task` triggers a mechanism for sending appropriate notifications to others. The user can also send e-mail from this view or set up a context-aware reminder. Finally, SCOUT provides a `Launch IDP` button to access the application related to this business process, if available on the Web. (IDP stands for Individual Development Plan.)

The presentation of tasks within the task list view (Figure 1) and the management of individual tasks (Figure 2) can change based upon contextual information. Our discussion here is based on a general notion of context; we will discuss the specific sources of context information and their derivatives in the next section. Two examples of the use of contextual data are (1) the availability of a person required to carry out the task and (2) context-aware reminders.

The availability of a person required to carry out a task is needed in order to schedule time for the task owner to work on the task. SCOUT makes use of

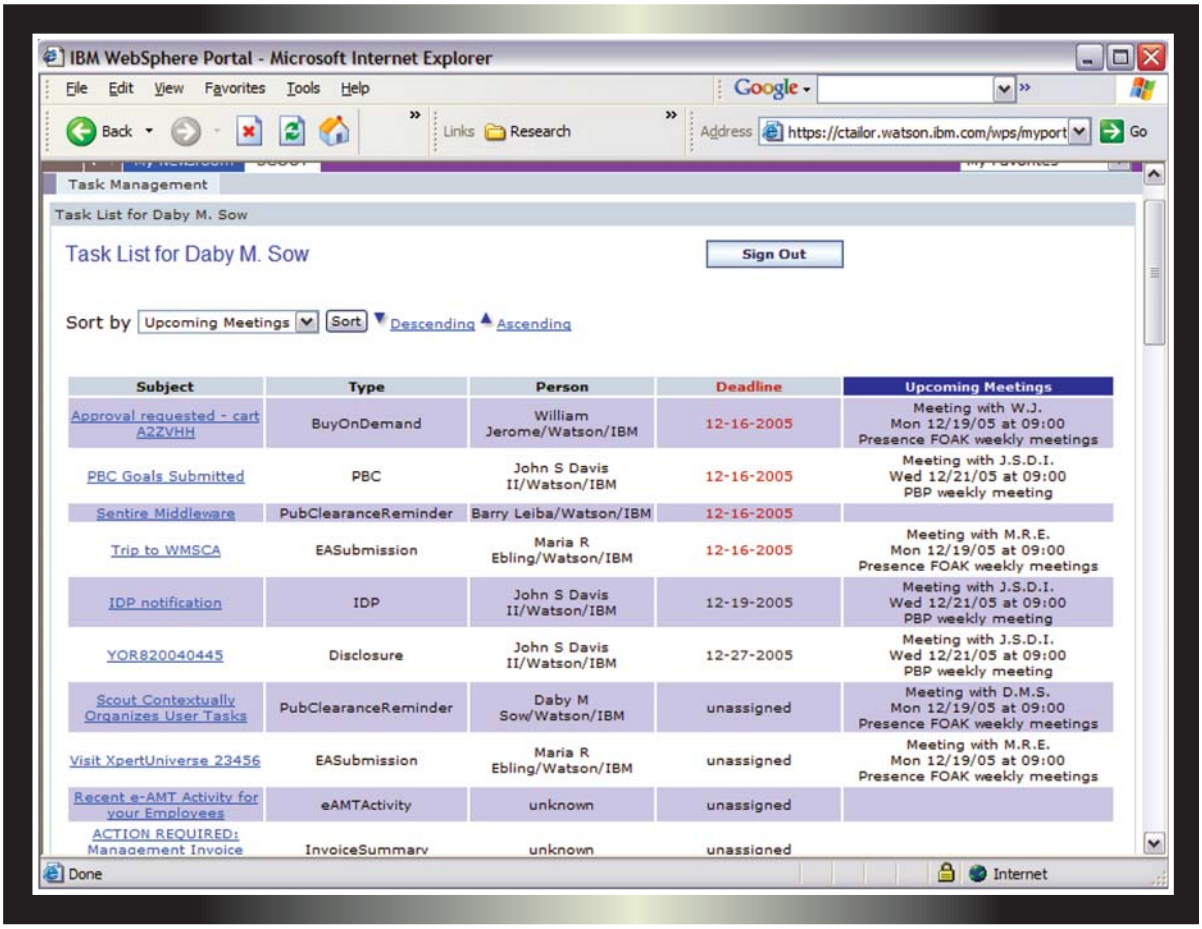


Figure 1
SCOUT portal application: task list view

several contextual data sources to infer a person's availability: calendar data, instant messaging (IM) status, phone status, and so on.

Contextual information is also used for context-aware delivery of messages to a task owner. Both proximity-based and availability-based reminders make use of contextual information. If a pending task requires the task owner to visit a particular location, the owner can arrange for a proximity-based reminder that will send the user a notification when he is in the vicinity of the location (e.g., remind the user to pick up a package from the mail room when the user is leaving the cafeteria which happens to be nearby). Similarly, if a pending task requires consultation with a colleague, the task owner can arrange for a reminder to be sent when the colleague becomes available or collocated (in the same room). In our current implementation, notification messages are delivered by means of Short Message Service (SMS).

E-mail client application

The SCOUT e-mail client application is integrated into the Lotus Notes* e-mail client used by a majority of the employees in our organization. The tasks identified by SCOUT are entered into the to-do list of the e-mail application. In addition, these tasks are shown in the calendar. The decision to show the to-do entries in the calendar is based upon the fact that the to-do feature of the e-mail application is not commonly used, whereas the calendar feature has an extremely high adoption rate within our organization.

Figure 3 shows a screen capture of the integrated Lotus Notes view of the to-do list. In this view the user can inspect the pending tasks and arrange and

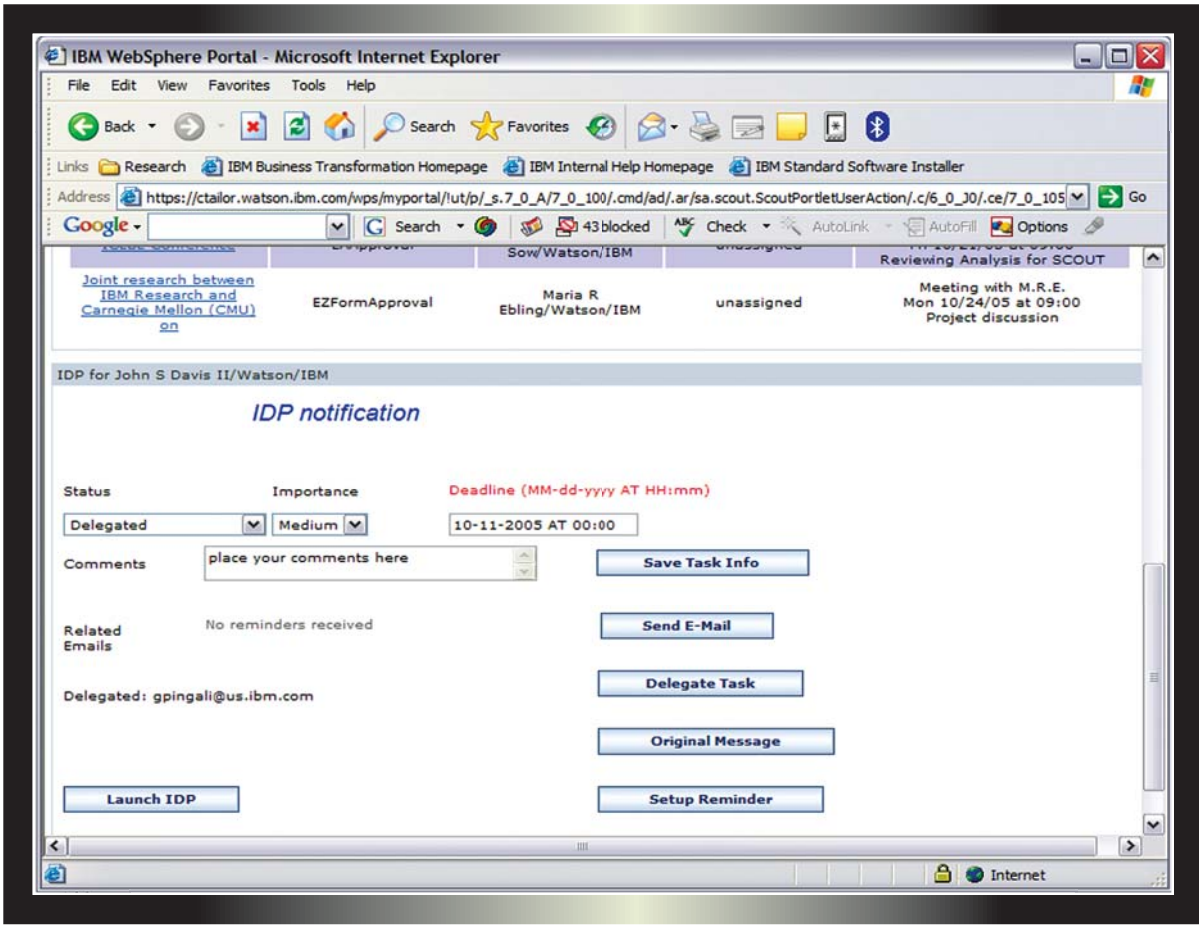


Figure 2
SCOUT portal application: task detail view

prioritize them according to several automatically extracted attributes, such as the due date, type, and subject. Figure 3 shows a view of tasks organized by type (Lotus Notes uses the term *category* instead of *type*). For example, the fourth task in category `EASubmission` represents a travel expense submission with subject `Hotel Bill WMCSA 2006` and deadline `02/07/2006`. The status of this task is shown to be `In progress`, and the task has been delegated to `John S. Davis II`. In addition, the entry shows that John has accepted the assignment.

Clicking on the subject link of a task opens up the task detail view, as shown at the bottom of Figure 3. The user can use standard Lotus Notes task management features to set the importance rating of the task and edit its deadline and status. In addition, the user may also delegate a task to a colleague, as discussed earlier. As can be seen in Figure 3, this interface supports a number of the metatasks presented earlier, including the prioritization of tasks, the monitoring of task status, the notification regarding partially completed tasks, and the delegation of tasks through explicit reassignment.

The e-mail client interface does not support the context-aware features provided by the SCOUT portal. This is because changes to the standard corporate mail template are difficult to deploy. Without such changes, context-aware features simply cannot be integrated into the e-mail client interface. The e-mail client application runs on the user's machine, which represents another important difference between the Web portal application and the e-mail client application. Because all processing is done on the user's machine (rather than on a server), this implementation has the benefit of improved security and privacy.

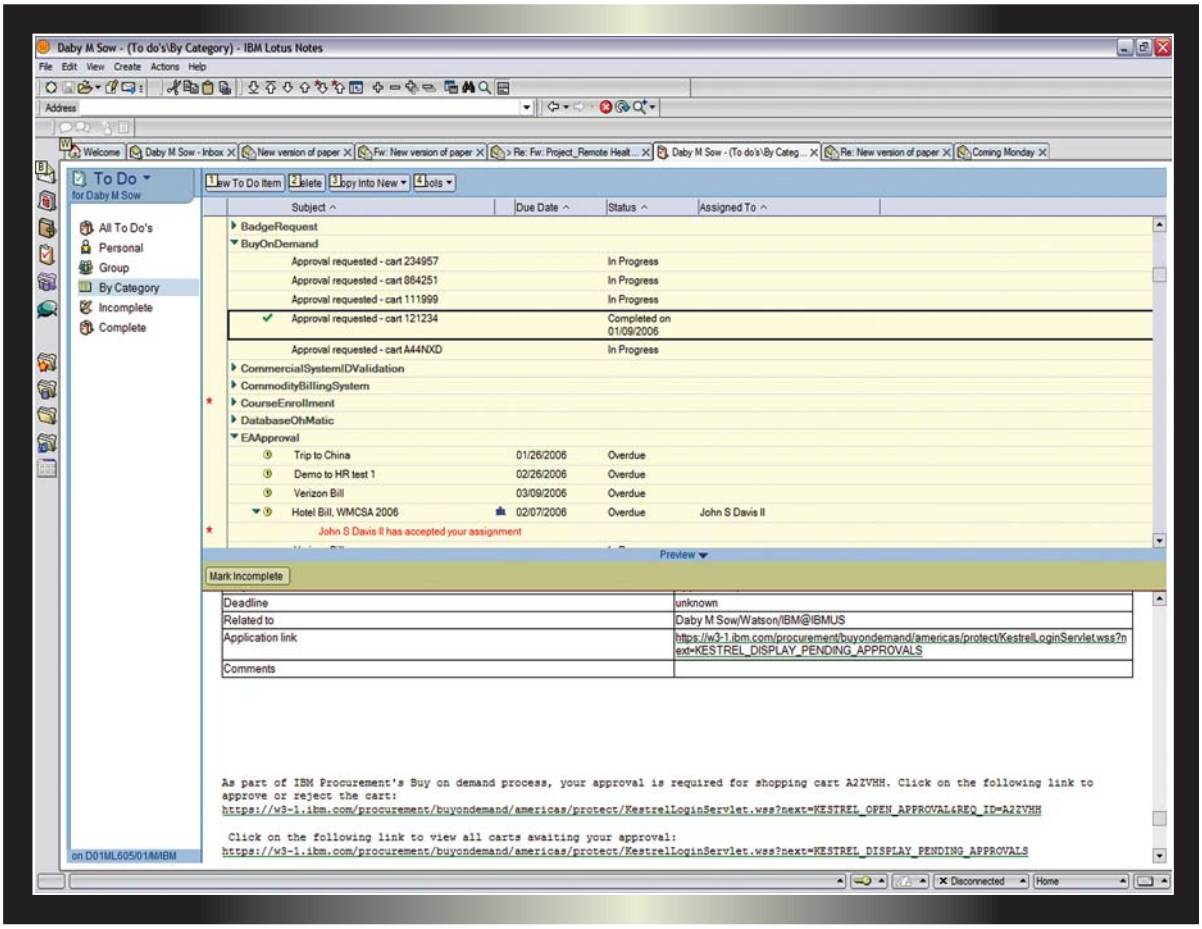


Figure 3
SCOUT e-mail client application: integrated Lotus Notes view of the to-do list

CONTEXT DATA SOURCES

The organization of a user's outstanding tasks should adapt to changing contextual attributes of the user and of any other individuals associated with the task. An example of adaptation based on the user's own context is modifying the task list according to the user's calendar. Examples of adaptation based on the contextual state of other individuals are proximity-aware reminders (e.g., when a pending task requires a conversation with the manager, alert the user when both are in the same room) or availability-aware reminders (e.g., when a travel reimbursement request requires a conversation with a colleague, alert the user that the individual has become available by highlighting the task entry in the task list view).

Most of the research on context-aware computing implicitly assumes the deployment of infrastructure to sense and collect context. It is sometimes hard to justify the capital cost of either the deployment of necessary sensors or the retooling of existing IT infrastructure components. Accordingly, our focus has been on identifying the dynamic user attributes that are already available and accessible. Contextual attributes can be classified as either raw or derived. *Raw* context refers to attributes obtained directly from external infrastructure components, such as sensors or software, and typically includes information such as a user's location, calendar entries, or IM status. *Derived* context refers to higher-level user attributes obtained by composing, or fusing, raw contextual data. For example, a person's availability (or willingness to be interrupted) might be deduced from a combination of calendar information, IM status, and phone status.

Context may also be classified as either physical (referring to physical user attributes such as location) or virtual (referring to attributes that exist

only within the IT infrastructure, such as the number of open IM sessions). As shown in *Figure 4*, the raw-derived dimension and the physical-virtual dimension can be viewed as orthogonal axes in the space of context sources. In the rest of this section we discuss the way we use a number of context sources in SCOUT.

Raw context sources

In this section we describe the following raw context sources used in SCOUT: calendar, IM presence, location, and phone status.

Calendar

In an enterprise environment, the calendar provides a rich source of context information. A SCOUT component known as the calendar adapter interface to a calendar server retrieves and parses calendar information. It gathers information on meeting and appointment events, which includes a list of attendees and the location, time, and topic of the meeting. It is worth noting that not only does calendar information provide insight about a user's current state (e.g., located in room 1401) but also about the user's likely future activity (e.g., scheduled for an off-site meeting in New York in an hour). The calendar information is particularly useful in determining if certain individuals are busy on specific tasks and should thus be free from interruption.

IM presence

Given the popularity of IM-based collaboration and communication within the enterprise, a user's current IM status also provides a very useful form of context. In particular, we use IM APIs for both synchronous and asynchronous retrieval of a user's presence-related events (e.g., online, offline, away from the computer, or in do-not-disturb mode).

Location

The user's location may be obtained in a number of ways. We focus here on the location within a building in which we have deployed an active badge infrastructure. The technology consists of radio frequency identification (RFID) readers installed at selected points (e.g., entrance and exit to the cafeteria) that detect when card-shaped badges (worn by individuals) are in the vicinity. Indoor location may also be inferred from wireless local area network (WLAN) connections.

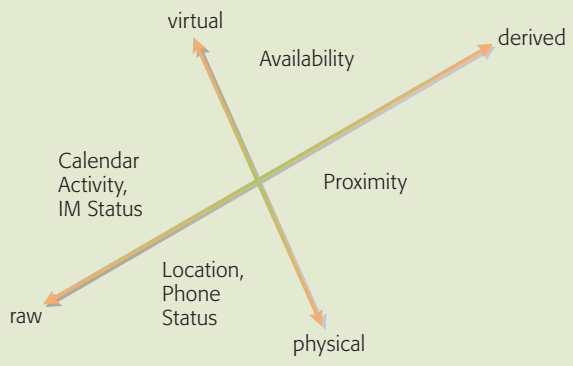


Figure 4
The space of context sources

Phone status

The activity status of the user on a phone (e.g., on/off hook, the identity of the callee) is retrieved through protocol-specific mechanisms. For Voice over Internet Protocol (VoIP) traffic based on the Session Initiation Protocol (SIP) protocol, we use special SIP OPTION messages to interrogate the SIP server or the end device about its status. As VoIP phones become universally deployed and SIP technology matures, presence-based notification mechanisms will provide a scalable solution for obtaining the phone status of individuals or groups.

Derived context sources

Based on the preceding raw sources of context, we derive the following contextual attributes: proximity and availability.

Proximity

The typical use of this attribute involves detecting when two people are in the same vicinity within an office building. We refer to these people as being collocated when they are in the same office or the same public space, such as a cafeteria. In SCOUT the detection of proximity relies on an active badge system. Proximity is useful, for example, when a user is to be reminded at the opportune time of an outstanding task that requires a face-to-face meeting with another party.

Availability

At present, availability is by far the most frequently used derived contextual attribute. Availability is an extension of the current network-centric notion of presence, in that the latter refers to the ability to communicate (e.g., whether an IM can reach the user), whereas the former refers to the user's

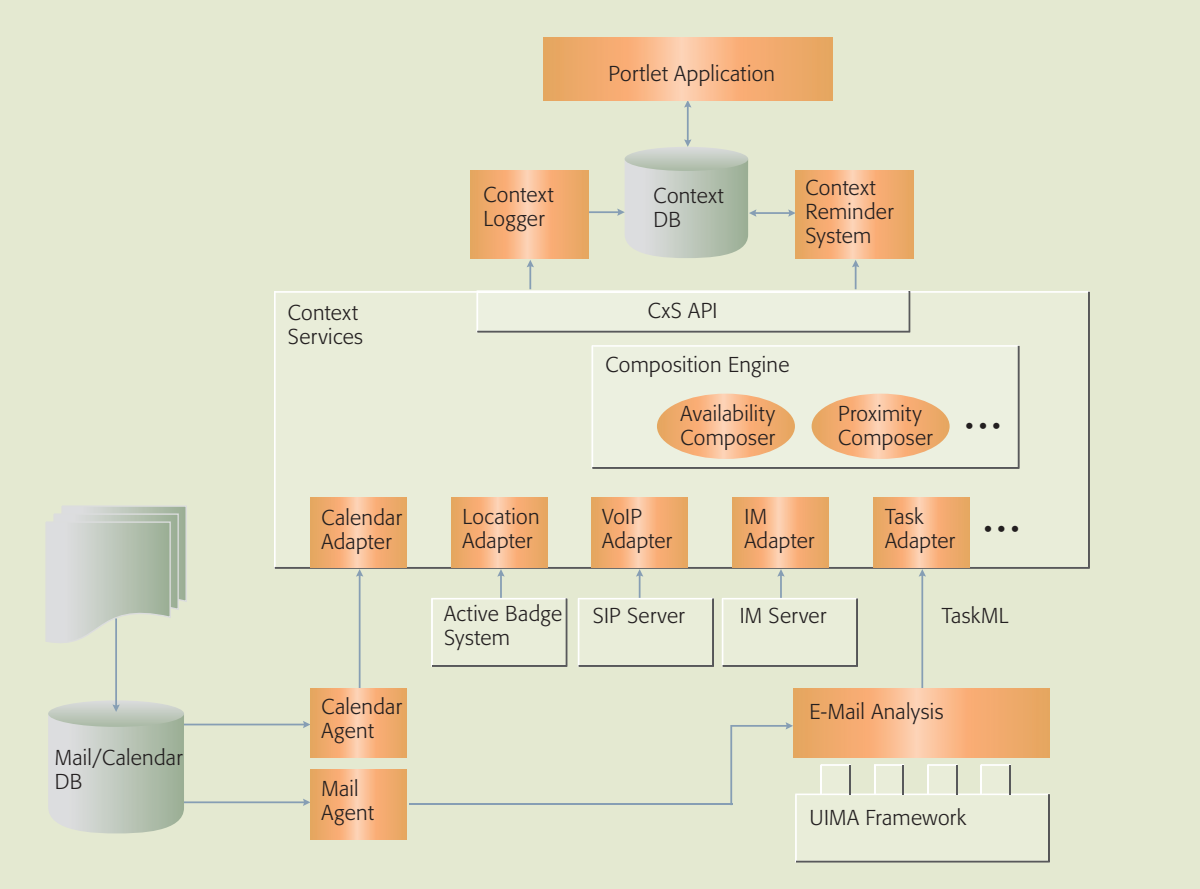


Figure 5
SCOUT system architecture

availability to actually participate in a communication session. In addition, whereas presence is a binary-valued attribute (a user is either connected or disconnected from IM), availability is multidimensional; a user may be available for IM but only with company employees, or the user may be available for voice calls but only with family members. Automatic, context-driven determination of availability (see, for example, References 19–22) is particularly appealing as it relieves the user of the responsibility of configuring the access rights granted to other users. We currently determine an individual's availability by combining their IM presence, voice status, and calendar information.

The addition of proximity further refines the notion of availability. For example, a user is considered to be attending a scheduled meeting (according to the calendar entry) only if SCOUT determines that the user is either located in the meeting room specified in the calendar entry or has dialed into the teleconference specified in that entry.

SYSTEM ARCHITECTURE

Figure 5 illustrates the architecture of the full-featured version of SCOUT. SCOUT has four main components: the Mail Agent component for accessing e-mail messages, the E-Mail Analysis module, the Context Services module for aggregating context data, and the Portlet Application module which provides the client interface. The SCOUT architecture is extensible so that a variety of client interfaces, e-mail systems, context sources and e-mail analysis modules can be supported.

The Mail Agent component serves as a translator that hides from SCOUT the syntactic details of proprietary e-mail systems (e.g., Lotus Notes, Microsoft Outlook**). The Mail Agent component reads e-mail messages that are stored in the third-

party mail database and translates each e-mail message into an EmailML document. EmailML is a simple XML description of an e-mail message that consists of the following attributes: from-address, to-address, subject, body, time sent, and unique ID. The unique ID is assigned by Mail Agent.

The output of the Mail Agent is consumed by the E-Mail Analysis module. This module is built on the IBM UIMA framework.[7] It maps incoming EmailML documents into TaskML documents. Details on this transformation are provided in the section "E-mail task analysis."

We use Context Services (CxS) to aggregate all context data as well as TaskML instances for delivery to the client interface.[23,24] CxS is designed for general support (API) of context-aware applications. The CxS architecture includes three major parts: a set of *adapters*, a *composition engine*, and an *application interface*. These parts are described in more detail below.

Adapter

Each type of context data that CxS supports is associated with an adapter. Figure 5 shows five adapters associated with data sources currently supported by CxS: calendar, location, IM, VoIP, and task. Each adapter retrieves or receives data of a given type from its context source. The retrieval of data occurs according to a specified schema, whereas a specified communication mechanism determines the way in which data will be sent to or retrieved by the adapter. For instance, the task adapter supports the TaskML schema. This adapter is essentially a Web service receiving TaskML documents encapsulated in Simple Object Access Protocol (SOAP) requests issued by the E-Mail Analysis module.

Composition engine

The CxS composition engine allows different sources of context to be aggregated. Specifically, we implemented availability and proximity composers, as shown in Figure 5. These derived context attributes were discussed in the section "Derived context sources." By delivering TaskML data through CxS (instead of uploading it directly into Context DB), we allow future applications to use the compositional facilities of CxS to aggregate TaskML documents with context data.

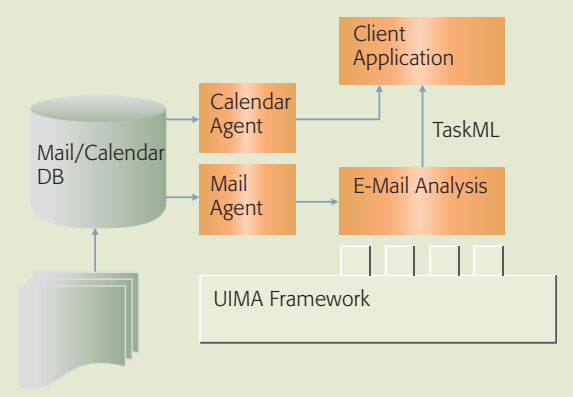


Figure 6
SCOUT architecture of the stand-alone implementation

Application interface (API). The client application of SCOUT sits on top of CxS. It uses the CxS API shown in Figure 5 to receive TaskML instances as well as relevant context data. Our context-aware reminder system is also a component sitting on top of CxS. SCOUT can post reminders to the reminder system based upon user preferences and the received TaskML.

Applications interface with SCOUT in two ways. They can access a database (context DB) where TaskML and calendar events are stored. This database is managed by a Context Logger module that uses the CxS API to subscribe to TaskML and calendar events. Applications can also retrieve contextual data about users (i.e., current location, current availability) by using the CxS API directly.

SCOUT can be configured to exclude context features when sensor resources are not available. The context features discussed in the previous section are only available in the SCOUT portal implementation. ***Figure 6*** shows a simplified architecture for the stand-alone implementation (e-mail client application). In this simplified version, the output of the calendar agent and the E-Mail Analysis module are directly consumed by the client application.

E-MAIL TASK ANALYSIS

SCOUT automates the task awareness aspect of the task management life cycle by analyzing e-mails in three stages, as illustrated in ***Figure 7***. The first stage, carried out by the task identification module,

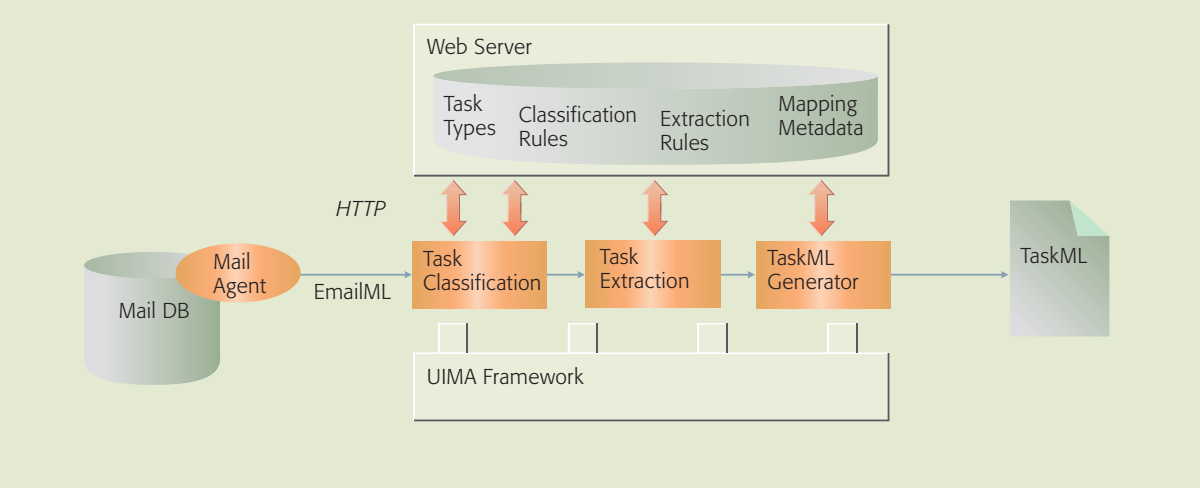


Figure 7
Architecture of the E-Mail Analysis module

involves identifying the task associated with an e-mail message and then labeling the message accordingly. In the second stage, the task extraction module further analyzes the labeled e-mails to produce a set of task attributes. Finally, the TaskML Generator module uses the set of task attributes to map the incoming e-mail into a TaskML document. The execution of each of these modules relies on sets of rules. These rules define the business processes supported in the system. They are maintained by an administrator on a Web server, as shown on the top part of Figure 7. E-Mail Analysis modules periodically poll this Web server to make sure that they are using the latest sets of rules published by the administrator. The current polling frequency is a parameter that is currently set to 180 minutes.

The e-mail analysis process is essentially a classification problem that can be performed either through statistically-based machine-learning algorithms or by a rule-based system that uses human-defined rules. Both approaches implement a translation function that maps e-mail messages to tasks. In the machine-learning approach, a machine-learning algorithm is determined through the use of training examples that consist of e-mail message and task pairs. In the rule-based approach, the mapping function is determined by human experts, who enter the rules into the system.

After reviewing a number of business-process-generated e-mails, we decided that a rule-based approach was the best method for performing e-mail analysis. Our decision was based on the deterministic nature of machine-generated e-mails, which allows humans who have knowledge of business process domains to easily create the analysis rules. Looking at e-mail samples sent by a business process often reveals numerous strings of text that are invariant throughout all these e-mail samples. These strings define a unique signature that may be encapsulated in very simple static rules that determine the e-mail-to-task translation function.

SCOUT rule-based analysis

The process of defining rules for the SCOUT rule-based engine starts with collecting samples of e-mails sent by each registered business process. For each business process, all sample e-mails are then aligned to identify a unique signature that is used to define a regular expression representative of the set of all task e-mails sent by this process. Such regular expressions are used to define these rules. For example, the following regular expression is currently used to identify all task e-mails sent by the Course Enrollment business process:

Subject: [ ] * [*eE*]*nrollment*[ ] + [*cC*]*onfirmation*[ ] + [*a–zA-Z*,@0–9] * [ ] + [*cC*]*ourse*[ ] + [*A-Za-z*0–9] + [ ] + [*cC*]*lass*[ ] + [*a-zA-Z*0–9] +

The task identification module is a UIMA annotator that uses rules to assign task types to each e-mail that it analyzes. The task extraction module is also a UIMA annotator in which e-mails tagged by the task

identification module are further analyzed to extract task attributes. This module uses the task type identified during the identification stage to load the appropriate set of rules defining how the e-mail should be parsed to identify relevant task attributes. This set of rules consists also of regular expressions that are used to annotate different parts of the e-mail corresponding to the different task attributes.

The annotations produced by the task extraction module are then fed into the TaskML Generator module which orchestrates the generation of the TaskML document. It uses user-defined preferences to assign values for task importance and deadline attributes when these are not provided in the e-mail. The TaskML generator module uses two additional UIMA annotators to validate the received annotations. The first one validates person names against a Lightweight Directory Access Protocol (LDAP) enterprise directory, and the second one formats dates and times representing task deadlines.

Evaluating the SCOUT rule-based approach

We have performed a series of tests to estimate the performance of the SCOUT rule-based approach. For our tests, we have registered 33 business processes that generate e-mails in the system. We then applied our analysis module on a corpus of 2,269 e-mails, corresponding to six weeks of machine-generated e-mails received by a real user.

As with any expert system, there are two types of classification errors that can occur in this system: false alarms and misses. In the case of a false alarm, SCOUT wrongly tags an e-mail as corresponding to a business process task, which indicates that the rules fail to filter out certain unsuitable e-mails. A miss occurs when SCOUT fails to identify an e-mail as a business process task; in this case the rules may be too narrow. In practice, misses are likely to be caused by changes in the format of some machine-generated e-mails.

Table 1 reports the results of this study. We note a very high classification accuracy of 0.9996 (obtained by subtracting the error rate of 1/2269 from 1) and a very good precision for detected tasks of 0.9583 (obtained by dividing the number of correctly predicted BP tasks by the number of BP tasks in the corpus). These results support our initial hypothesis on the performance and robustness of the rule-based approach for the problem at hand. This tendency of missing very few tasks was further verified with feedback obtained from end users during user studies described in the section "Application evaluation." In fact, only one user reported misses, and in this case, all such errors were due to an update to the format of the e-mails sent by our company's travel-expense reimbursement process.

Table 1 Summary of the SCOUT rule-based classification result on a real in-box.

Time window	05/12/06–06/28/06
BP tasks in corpus	24
Correctly Predicted BP Tasks	23
Non-BP Tasks	2245
Tasks identified by SCOUT	24
Misclassifications	1

A hybrid approach for improved performance

Although the rule-based classification method achieved a very high rate of accuracy, the nature of business process e-mails is such that even small error rates are unacceptable. For example, failure to properly detect a task requesting a manager to approve an employee travel-expense report may result in suspension of the employee's corporate credit card—an intolerable situation. To address this problem, we developed an error detection system that uses both rule-based and machine-learning algorithms and allows a SCOUT administrator to detect when rules need to be updated. This system, shown in *Figure 8*, consists of the SCOUT task identification module, working in conjunction with a machine-learning document classifier.

The SCOUT error detection system is based on two premises that are characteristic of the business process e-mails which we are targeting. First, as demonstrated by our validation study, a rule-based system is able to achieve a very small error rate, but the rules are brittle with respect to business processes that are likely to change over time. (The machine-generated business process e-mails are usually changed over time due to various management decisions—unfortunately, such changes are often not communicated to the company as a whole. In addition, a few errors are typically present due to judgment lapses by the rule writers.) Second, a

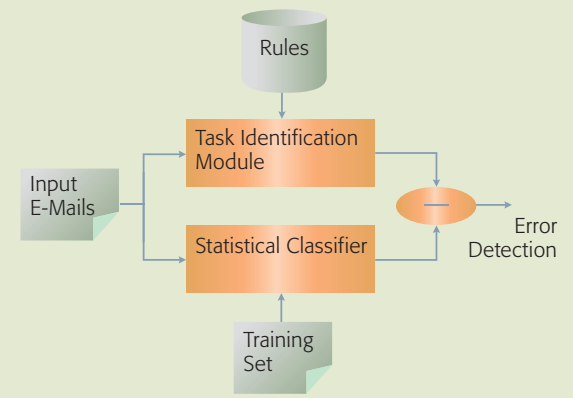


Figure 8
SCOUT error detection mechanism

machine-learning approach suffers from a small but not insignificant false alarm rate; often such false alarms result from messages related to a business process, but the messages do not contain a task for the recipient to complete. The result is that both approaches suffer from errors when used alone. Given these characteristics, we designed our error detection system to use the machine-learning and the rule-based approaches in parallel, taking advantage of their respective strengths and, at the same time, minimizing errors in handling changing business processes.

As shown in Figure 8, the outputs of the task identification module (the rule-based algorithm) and the statistical classifier module (the machine-learning approach) are compared. A discrepancy in their outputs is a hint that either the task identification module or the statistical classifier module may have misclassified the corresponding document. In the former case, the error may be either a false alarm or a miss, and it requires the administrator to update the rule set accordingly. In the latter case, the administrator needs to update the training set of the statistical classifier to include the current test case and retrain it.

In the tests we performed to evaluate the performance of our error detection scheme, we used the statistical classifier proposed by Johnson et al.[25] We selected this text categorization technique for its efficiency, in terms of both predictive accuracy and computational complexity. We trained this statistical classifier with a corpus of 92 e-mail messages. This training set is composed of 72 machine-generated e-mails from eight different business processes and 20 human-generated e-mails selected randomly from a real in-box. Two of these eight business processes changed during the lifetime of the SCOUT project. The format of the messages that they produced was updated. For our training set, we used the old mail format for these two processes. Each of these 92 e-mail messages were then manually labeled with their corresponding task type. Human-generated e-mails were labeled "unstructured." Finally, in order to avoid overfitting on the training data, all the messages in this training set were also made anonymous by replacing certain fields with random characters. ("Overfitting" refers to the process of training a learning algorithm in which the learner adjusts to inconsequential details in the training data, which leads to errors on unseen data.[26])

The experimental testing set used to evaluate this error detection scheme is composed of 119 e-mails. This experimental corpus was synthesized from real human-generated e-mails extracted from a real in-box and simulated business process e-mails with errors intentionally introduced to test the capabilities of this error detection scheme. More specifically, 89 of 119 e-mails were generated from the eight different business processes used in the generation of the training set. The remaining 30 were human-generated. We used a mixture of old and new message formats for the two processes that changed during the lifetime of the project.

On the experimental testing set described above, 40 errors were detected. Interestingly, the statistical classifier was able to correctly classify all e-mails sent by the two processes that changed during the lifetime of the project, despite the message format updates. On the other hand, the task identification module failed to correctly classify all the messages in the new formats, resulting in 20 error detections (because the rules were based on the old e-mail format). The remaining 20 errors were false alarms due to misclassifications from the statistical classifier. Indeed, it wrongly detected expense account tasks on e-mails that were supposed to be classified as unstructured. Adding these test cases to the training set significantly reduced this false alarm rate.

APPLICATION EVALUATION

We describe here a user study we carried out in order to evaluate the effectiveness of SCOUT in the real world. The study, which covered both the

SCOUT portal application and the SCOUT e-mail client application, was performed in three phases. In the first phase, a predeployment survey was sent to the participants to gauge their perceptions regarding e-mail-based tasks. Following the collection of the predeployment survey results, we asked the participants to use SCOUT for three to four weeks. Finally, in the last phase, participants were asked to assess the strengths and weaknesses of SCOUT by means of a structured exit interview.

The 14 study participants were split in two groups. A first group of seven users were asked to use and evaluate the SCOUT portal application for several weeks. These participants were not given the ability to set preferences for task deadlines and urgency ratings. When these attributes were not available in the body of the e-mails, they had to assign these attributes to their tasks manually through the portal interface. The seven participants in the second group were asked to use and evaluate the SCOUT e-mail client application. This study included 33 business processes registered in the system.

Overall, the SCOUT application was well received by our user population. ***Figure 9*** summarizes the overall user perception of SCOUT in a histogram where the X axis represents the user ratings on a Lickert scale of 1 (extremely useless) to 5 (extremely useful). The Y axis represents the number of participants for a given rating. There are two series of values represented on this figure. The green histogram refers to ratings obtained when users were asked, "How useful did you find SCOUT overall?" The orange histogram refers to ratings obtained when users were asked, "For the tasks that it managed, how useful was SCOUT to you?" In both series, almost all of the respondents gave rating equal to or above 4.0.

Three participants rated the overall usefulness of the system with a score of less than 4. These participants were all users of the SCOUT portal application who complained about the lack of integration of SCOUT with their e-mail client. These users would rather deal with just one client application to manage both their e-mails and their tasks. Unfortunately, a Web-based e-mail solution that we could integrate with our portal is currently not supported.

Most users found the following SCOUT functionalities very useful to them:

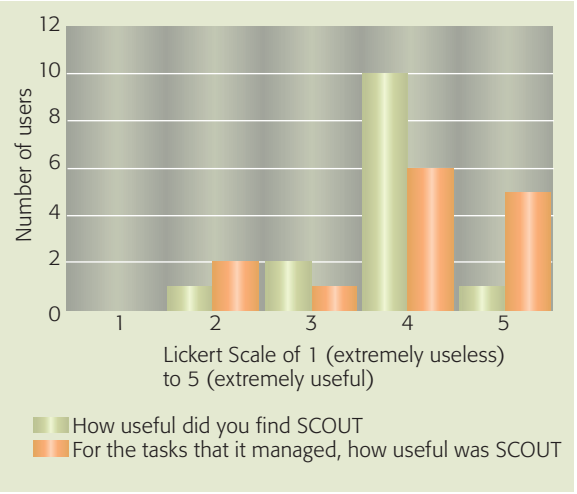


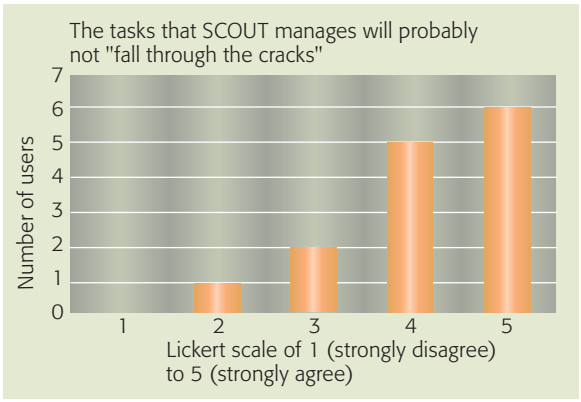
Figure 9
Overall user perception of SCOUT

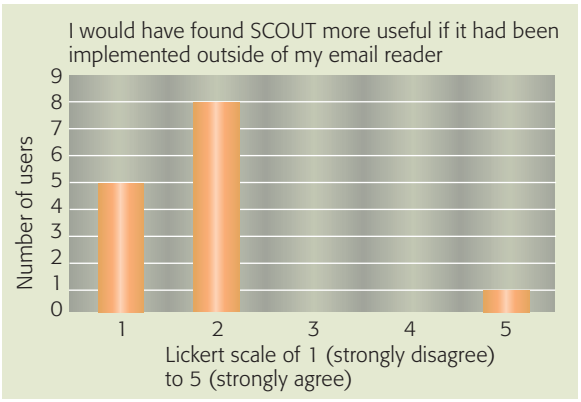
- Automatic tracking and extraction of tasks from e-mails
- Creating a task list in which tasks can be executed in batch
- Sorting tasks into folders
- Sorting tasks by various attributes

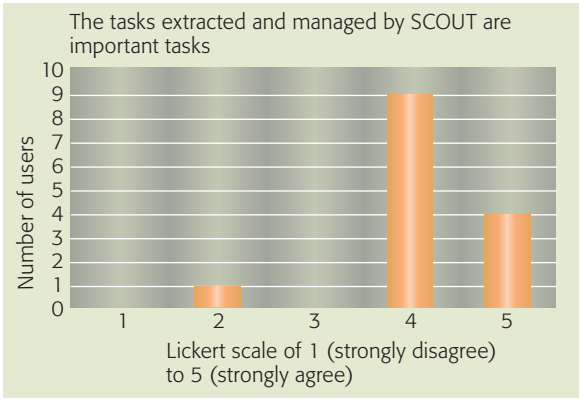
To further assess the usefulness of the tool the participants were asked to answer a number of detailed questions. A summary of the answers obtained are shown in ***Figure 10***. Users acknowledged occasionally having tasks "fall through the cracks" before using SCOUT, with an average rating of 4.3. The statement "The tasks that SCOUT manages will probably not fall though the cracks" drew even stronger agreement, with an average rating of 4.4. Agreement (with an average rating of 4.1) with the statement "The tasks extracted and managed by SCOUT are important tasks" clearly shows the utility of this task management system, despite being restricted to handle machine-generated e-mails only.

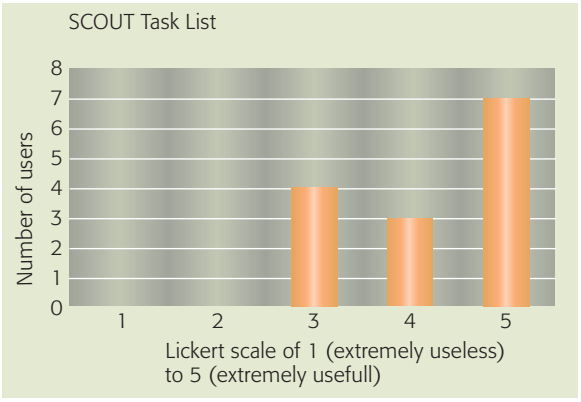
The exit interview also gathered end-user feedback on the utility of the following specific features within SCOUT:

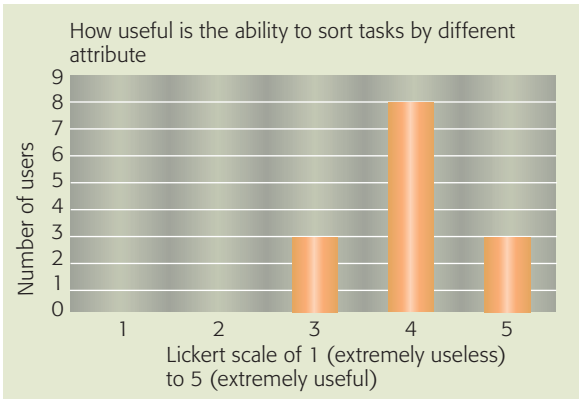
Task organization—Participants found the ability to view tasks in a list (average rating of 4.2), the ability to sort the list by different attributes (average rating of 4.0), and the ability to file task e-mails into specific folders (average rating of 3.9) to be quite

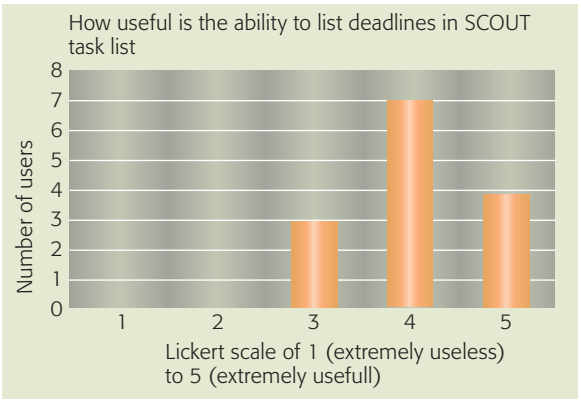


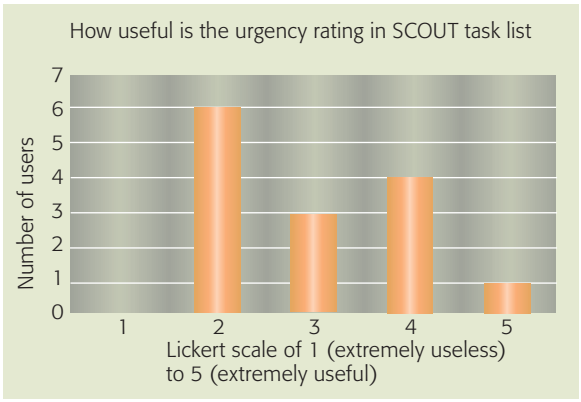












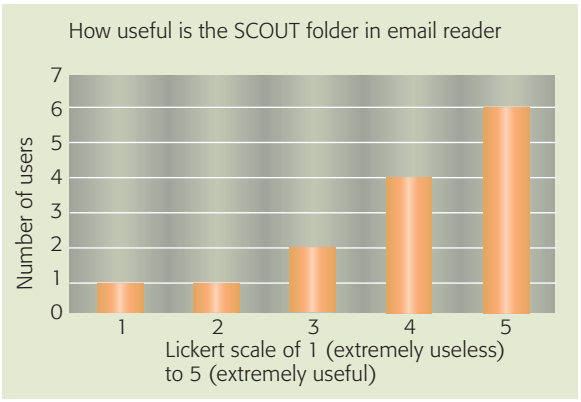


Figure 10
User perception of SCOUT features

useful. Even the users of the e-mail client application, with a smaller set of attributes that they could use to sort their tasks, found this aspect of the application to be quite useful.

Task attributes—The listing of deadlines in the task list received an average rating of 4.0. For SCOUT portal users, because this feature was not automated, it was of somewhat lesser value. A user of the SCOUT portal application stated that the manual entry required for business processes that do not explicitly state task deadlines within the generated e-mails makes this feature less useful. In contrast, the SCOUT e-mail client application users liked this feature, especially with deadlines being linked with task reminders placed in their calendar. The manually assigned urgency rating in SCOUT was rated low (average rating of 3.0). Participants asked for a fully automated urgency rating system, without requiring users to define and manage their preferences.

Task delegation—The ability to delegate a task within SCOUT received an average rating of 3.1. Participants who assigned this feature a low rating commented that although SCOUT manages the notifications of task delegation status exchanged between the delegate and the sender, there is no automated transfer of authority to perform the task inside the business process. Moreover, these participants often do not delegate the tasks currently managed by SCOUT.

CONCLUSIONS

The enormous quantity of daily e-mails coupled with the high degree of importance associated with some of these messages is turning e-mail management into a challenging process. It has become essential to be able to identify and manage tasks for which the recipient of the e-mail is responsible. We designed SCOUT to automate this task management process by automatically extracting tasks from in-box messages by using rule-based e-mail analysis. The population targeted as potential users were managers who receive daily a large number of machine-generated e-mail messages.

SCOUT is currently in use by tens of users. It has been well received and has generated positive end-user feedback. A participant wrote to us that his use of the SCOUT portal application enabled him to discover pending tasks in his in-box that he missed. A user of the SCOUT e-mail client application was pleasantly surprised to discover an important and forgotten task on his calendar.

Given the fact that most e-mail is unstructured, a future extension of SCOUT will incorporate an e-mail analysis engine that can extract tasks from unstructured messages. This extension will allow tasks to be identified within messages generated by humans. Initial results on the use of machine-learning techniques to identify such e-mails are promising. Our research agenda includes the development of mechanisms able to extract task attributes from unstructured e-mails.

Another extension of SCOUT may focus on clustering tasks belonging to a user-defined activity (e.g., all the tasks related to a user's last trip to Paris). This capability may improve the productivity of end users by allowing them to focus on tasks within an activity, with a common thread that they control.

SCOUT is scheduled to be widely deployed in the near future. This large-scale deployment presents us with a unique opportunity to run a large-scale user study and refine our assessment of the strengths and weaknesses of our approach.

ACKNOWLEDGMENTS

We thank Guruduth Banavar and Michael Greenwood for their invaluable contributions. In addition, we thank all the SCOUT users, with a special mention to the study participants, for helping us to improve this system.

CITED REFERENCES

1. S. Whittaker and C. Sidner, "Email Overload: Exploring Personal Information Management of Email," *Proceedings of the CHI '96 Conference on Human Factors in Computing Systems*, Vancouver, BC, Canada, April 13–18, 1996, pp. 276–283.
2. S. Whittaker, T. P. Moran, and S. P. Farrell, "Why Email is Not Enough: Combining Communication and Shared Representation to Support Activity Management," *Proceedings of the 9th European Conference on Computer-Supported Cooperative Work (ESCSW 2005), Workshop*

on Computer Support for Human Activity, Paris, France, September 17, 2005 (paper available from authors).

3. E. V. Wilson, "Email Winners and Losers," *Communications of the ACM* **45**, No. 10, 121–126 (October 2002).
4. M. Eisenstadt, "Eight years of email stats, pass 1," February 11, 2005, http://www.corante.com/getreal/archives/2005/02/11/eight_years_of_email_stats_pass_1.php.
5. T. P. Moran, "Activity: Analysis, Design, and Management," in *Theories and Practice in Interaction Design*, S. Bagnarx and G. Crampton Smith, Editors, Erlbaum Press, Mahwah, NJ (2006).
6. D. Sow, M. Ebling, R. Lehmann, J. Davis, and L. Bergman, "SCOUT Contextually Organizes User Tasks," *Proceedings of the IEEE International Conference on e-Business Engineering, ICEBE 2005*, Beijing, China, October 12–18, 2005, pp. 94–101.
7. D. Ferrucci and A. Lally, "UIMA: An Architectural Approach to Unstructured Information Processing in the Corporate Research Environment," *Natural Language Engineering* **10**, Nos. 3–4, 327–348 (2004).
8. R. Bekkerman, A. McCallum, and G. Huang, "Automatic Categorization of Email into Folders: Benchmark Experiments on Enron and SRI Corpora," CIIR Technical Report IR-418, University of Massachusetts, Amherst, MA (2004).
9. V. Bellotti, B. Dalal, E. Good, P. Flynn, D. Bobrow, and N. Ducheneaut, "What a To-Do: Studies of Task Management Towards the Design of a Personal Task List Manager," *Proceedings of the 2004 Conference on Human Factors in Computing Systems (CHI 2004)*, Vienna, Austria, April 24–29, 2004, pp. 735–742.
10. V. Bellotti, N. Ducheneaut, M. Howard, and I. Smith, "Taking Email to Task: The Design and Evaluation of a Task Management Centered Email Tool," *Proceedings of the 2003 Conference on Human Factors in Computing Systems, CHI 2003*, Ft. Lauderdale, Florida, April 5–10, 2003, pp. 345–352.
11. J. Gwizdka, "Reinventing the Inbox—Supporting the Management of Pending Tasks in Email," *Extended Abstracts of the 2002 Conference on Human Factors in Computing Systems, CHI 2002*, Minneapolis, Minnesota (2002), pp. 550–551.
12. S. Corston-Oliver, E. Ringger, M. Gamon, and R. Campbell, "Task-Focused Summarization of Email," *Proceedings of the ACL-04, Workshop on Text Summarization Branches Out, 42nd Annual Meeting of the Association for Computational Linguistics*, Barcelona (2004), http://acl.ldc.upenn.edu/acl2004/textsummarization/pdf/Corston.pdf.
13. P. Bennett and J. Carbonell, "Detecting Action-Items in E-mail," *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development on Information Retrieval (SIGIR '05)*, Salvador, Brazil, August 15–19, 2005, pp. 585–586.
14. R. Segal and J. Kephart, "MailCat: An Intelligent Assistant for Organizing E-Mail," *Proceedings of the Third Annual Conference on Autonomous Agents, AGENTS '99*, Seattle, WA, May 1–5, 1999, pp. 276–282.
15. M. Dredze, T. Lau, and N. Kushmerick, "Automatically Classifying Emails into Activities," *Proceedings of the 2006 International Conference on Intelligent User Interfaces, IUI 2006*, Sydney, Australia, January 29–February 1, 2006, pp. 70–77.
16. J. Shen, L. Li, T. Dietterich, and J. Herlocker, "A Hybrid Learning System for Recognizing User Tasks from Desktop Activities and Email Messages," *Proceedings of the 2006 International Conference on Intelligent User Interfaces, IUI 2006*, Sydney, Australia, January 29–February 1, 2006, pp. 86–92.
17. A. McCallum, "Information Extraction: Distilling Structured Data from Unstructured Text," *ACM Queue* **3**, No. 9, 48–57 (November 2005).
18. A. Tomasic, J. Zimmerman, and I. Simmons, "Linking Messages and Form Requests," *Proceedings of the 2006 International Conference on Intelligent User Interfaces, IUI 2006*, January 29–February 1, 2006, Sydney, Australia, pp. 78–85.
19. J. Fogarty, J. Lai, and J. Christensen, "Presence Versus Availability: The Design and Evaluation of a Context-Aware Communication Client," *International Journal of Human-Computer Studies*. **61**, No. 3, 299–317 (September 2004).
20. E. Horvitz and J. Apacible, "Learning and Reasoning about Interruption," *Proceedings of the 5th International Conference on Multimodal Interfaces, ICMI 2003*, Vancouver, Canada, November 5–7, 2003, pp. 20–27.
21. E. Horvitz, C. M. Kadie, T. Paek, and D. Hovel, "Models of Attention in Computing and Communications: From Principles to Applications," *Communications of the ACM* **46**, No. 3, 52–59 (March 2003).
22. S. Hudson, J. Fogarty, C. Atkeson, D. Avrahami, J. Forlizzi, S. Kiesler, J. Lee, and Jie Yang, "Predicting Human Interruptibility with Sensors: A Wizard of Oz Feasibility Study," *Proceedings of the 2003 Conference on Human Factors in Computing Systems, CHI 2003*, Ft. Lauderdale, Florida, April 5–10, 2003, pp. 257–264.
23. N. Cohen, P. Castro, and A. Misra, "Descriptive Naming of Context Data Providers," *Proceedings of the 5th International and Interdisciplinary Conference on Modeling and Using Context, CONTEXT 2005*, Paris, France, July 5–8, 2005, pp. 112–125.
24. H. Lei, D. M. Sow, J. S. Davis II, G. Banavar, and M. Ebling, "The Design and Applications of a Context Service," *Mobile Computing and Communications Review* **6**, No. 4, 45–55 (2002).
25. D. E. Johnson, F. J. Oles, T. Zhang, and T. Goetz, "A Decision-Tree-Based Symbolic Rule Induction System for Text Categorization," *IBM Systems Journal* **41**, No. 3, 428–437 (2002).
26. T. Mitchell, *Machine Learning*, McGraw Hill, New York, 1997.

Accepted for publication July 6, 2006.
Published online October 24, 2006.

Daby M. Sow
IBM Research Division, Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, New York 10532. (sowdaby@us.ibm.com). Dr. Sow is a research staff member at the Watson Research Center. His research interests range from theoretical problems in information theory and machine learning to middleware and application design in pervasive computing. He has published extensively and holds several patents in these fields. Before joining IBM, he spent the summer of 1998 at Lucent Technologies, Bell Laboratories, where he worked on high-rate video-coding systems. In the summer of 1999, he was with Philips Research Laboratories, working on computational resource scalability and scalable algorithms in image and video processing. He received a B.Sc. degree in electrical engineering from Université Laval, Québec, Canada, in 1994, and M.S. and Ph.D. degrees in

electrical engineering from Columbia University in 1996 and 2000. During his graduate studies, he was a member of the Multimedia Signal Processing Laboratory at Columbia University.

John S. Davis II
IBM Research Division, Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, New York 10532 (davisjs@us.ibm.com). Dr. Davis is a research staff member at the Watson Research Center. His research interests include pervasive computing, privacy, and the intersection of medicine and technology. Since joining IBM in 2000, he has participated in the implementation of research prototypes involving intelligent messaging, contextual pattern generation, context-aware reminders, and pervasive business workflow. He has numerous publications and several patents (filed or pending) in the field of context and location-aware computing. Before joining IBM, Dr. Davis held positions at G.E. Medical Systems and Hughes Space and Communications. He holds bachelor and doctorate degrees in electrical engineering from Howard University and the University of California at Berkeley, respectively. During his graduate work, Dr. Davis was a member of the Ptolemy research group in system-level design and electronic design automation.

Maria R. Ebling
IBM Research Division, Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, New York 10532 (ebling@us.ibm.com). Dr. Ebling is a research staff member at the Watson Research Center, where she manages the Privacy-Enabled Context Technologies department. Her group builds middleware to support context-aware computing with a focus on user privacy concerns and applicability to the health-care industry. She received a B.S. degree in mathematics from Harvey Mudd College and M.S. and Ph.D. degrees in computer science from Carnegie Mellon University. Her research interests are in distributed systems supporting mobile and pervasive computing, privacy, and human-computer interaction. Dr. Ebling is a member of the IEEE Computer Society and the Association for Computing Machinery.

Archan Misra
IBM Research Division, Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, New York 10532. (archan@us.ibm.com). Dr. Misra is a research staff member in the Distributed Computing department at the Watson Research Center. He received a B. Tech. degree in electronics and communication engineering from Indian Institute of Technology, Kharagpur, India in 1993, and M.S. and Ph.D. degrees in electrical and communication engineering from the University of Maryland at College Park in 1996 and 2000. At IBM, for the past five years, he has been working on middleware and protocols for pervasive and collaborative computing, including algorithms and architectures for collaborative applications over converged networks based on the Session Initiation Protocol (SIP), context-sensitive middleware technologies, high-performance wireless mesh networks, and data management for sensor-based applications. Before joining IBM, Dr. Misra worked at Telcordia Technologies, researching Internet QoS architectures and cellular mobility management protocols. He is a coauthor of papers that received the Best Paper awards at the ACM WOWMOM 2002 and IEEE MILCOM 2001 conferences. He currently chairs the IEEE Computer Society's Technical Committee on Computer Communications (TCCC).

Lawrence Bergman
IBM Research Division, Thomas J. Watson Research Center, 19 Skyline Dr., Hawthorne, NY 10532 (bergmanl@us.ibm.com). Dr. Bergman is a research staff member in the Software department at the Watson Research Center. He received a B.S. degree in physical sciences and zoology from the University of Maryland in 1977, and M.S. and Ph.D. degrees in computer science from the University of North Carolina at Chapel Hill in 1989 and 1993. He subsequently joined IBM at the Watson Research Center, where he has worked on computer graphics, image query, model-based application development tools, and desktop programming by demonstration. In 2000, he received an IBM Outstanding Innovation Award for his work on content-based image retrieval. Dr. Bergman is a member of the Association for Computing Machinery. ■