A Personal Email Assistant

Ruth Bergman, Martin Griss, Carl Staelin
Software Technology Laboratory
HP Laboratories Palo Alto
HPL-2002-236
August 22nd , 2002*

email, agents, machine learning, information retrieval

A key element of the CoolAgent Personal Assistant vision is the active management and use of personal, team and organizational information. The finding, filtering, composing, routing and information-triggered notification to a (mobile) user, adapted to the location, schedule, available appliances, tasks and other personal and team context is a key capability within the vision of an agent-based system for personal, professional and team activities. In this paper we report on the current status of a key element, the personal email assistant (PEA), which provides a customizable, machine-learning-based environment to support the activities of a major time sink of our daily lives - the processing of email. The system has been designed to be usable either with or without an agent-based infrastructure, and to be useful with a variety of email systems. In its current form, it leverages and augments the capabilities provided by Exchange and Outlook. It provides capabilities of: smart vacation responder, junk mail filter, efficient email indexing and searching, deleting, forwarding, re-filing, and prioritizing of email. A key contribution of our work has been to leverage high-quality open source components for information retrieval, machine learning, agents and rules to provide a powerful, flexible and robust capability.

* Internal Accession Date Only
Approved for External Publication

A Personal Email Assistant

Ruth Bergman[*], Martin Griss[#], and Carl Staelin[*]

[#]HP Laboratories, Palo Alto
1501 Page Mill Road
Palo Alto, CA 94304 USA

[*]HP Laboratories, Israel
Technion City
Haifa 32000, ISRAEL

August 2002

Abstract

A key element of the CoolAgent Personal Assistant vision is the active management and use of personal, team and organizational information. The finding, filtering, composing, routing and information-triggered notification to a (mobile) user, adapted to the location, schedule, available appliances, tasks and other personal and team context is a key capability within the vision of an agent-based system for personal, professional and team activities. In this paper we report on the current status of a key element, the personal email assistant (PEA), which provides a customizable, machine-learning-based environment to support the activities of a major time sink of our daily lives – the processing of email. The system has been designed to be usable either with or without an agent-based infrastructure, and to be useful with a variety of email systems. In its current form, it leverages and augments the capabilities provided by Exchange and Outlook. It provides capabilities of: smart vacation responder, junk mail filter, efficient email indexing and searching, deleting, forwarding, re-filing, and prioritizing of email. A key contribution of our work has been to leverage high-quality open source components for information retrieval, machine learning, agents and rules to provide a powerful, flexible and robust capability.

Keywords: email, agents, personal assistants, machine learning, information retrieval

1 Introduction

In this paper, we report on the vision, goals, status and key technical elements of a personal email assistant (PEA). A PEA is an application or suite of applications that proactively monitor and manage a user's email, to reduce the burden of the large volume of email that a typical user encounters today. So much email (both valuable and "junk") arrives each day that many users (individual and teams) are overwhelmed, missing important messages, responding late, forgetting to follow up and spending lots of time on rote email handling tasks. This is especially true in most larger, distributed organizations, which tend to use email as a primary communication medium. Users maintain many folders, are constantly moving mail between folders, archiving and retrieving mail, and searching for relevant mail.

Our vision of the PEA is as a key piece of a larger Personal Information Assistant (PIA) that indexes and manages content from several personal information sources, such as email, local files, and bookmarks/favorites, and provides a unified search over the personal, local, and global information. In turn, we envision the PIA as a component of a Personal Assistant (PA) that assists users in handling many tasks, involving email, calendars, context, and personal information. In addition to general information indexing and search, the toolkits and services associated with the PIA can

be used to respond to specific task-oriented queries, trigger actions based on the discovery of specific information or context changes (such as receipt of email, change in a web page or calendar, or completion of tasks), or construct task-oriented summaries of salient information for meetings, etc.

Intelligent agent projects to date are criticized for solving toy problems. The projects result in good demonstrations rather than in useful tools. The standard response is that the technology is ready, and the useful problems would be solvable if only we had a useful set of supporting web services. We, on the contrary, built the critical supporting services, namely contact and calendar services, on top of a high-quality, widely-used commercial system, namely Microsoft Exchange. With these services our PEA is truly useful.

The PEA supports these key email-related tasks:

1. prioritize – the PEA uses classification and rules to prioritize incoming messages. The user may then view email sorted by priority in Outlook.
2. filter – the PEA uses classification and rules to filter unwanted mail, such as "junk" mail.
3. index/retrieve – the PEA maintains an inverted index of all current and archived mail which the user may search with a Google-like interface.
4. refile – the PEA uses classification and rules to move messages to appropriate folders.
5. vacation response – a Vacation agent uses the contact and calendar agents to respond to incoming messages with appropriate vacation responses[1].

Some of these tasks are available in mail client applications today, e.g. rule-based refiling in Microsoft Outlook/Exchange, junk mail filtering in Outlook/Exchange, Netscape or Hotmail, machine learning [ifile; Shahami 1998], and index/retrieve in Microsoft Outlook. We are not familiar with any application that provides prioritization, nor of any application that combines all of the above capabilities.

The PEA also demonstrates how useful tasks are accomplished via agent interaction. We have architected the PEA and the PA on which it is based as a set of interacting agents and services. The use of a multi-agent architecture allows great flexibility in dunamically discovering, configuring and customizing the system as new services are added. The Vacation agent is a simple, but useful, example of leveraging information to reduce user burden. This agent responds to incoming messages with the standard "I am on vacation until… " message. Unlike the vacation response available in most email clients, this agent requires no user intervention. Finding a vacation event in the calendar automatically triggers the vacation agent behavior. Furthermore, the vacation agent uses some heuristics to avoid sending multiple vacation messages to the same person, customizing the response for different (types of) people, and responding to mailing lists.

The PEA is designed as a suite of customizable, extensible elements[2] that work together to process incoming and existing email using a flexible combination of

[1] This can be generalized to create responses or invoke actions that are customized to the sender, message content and situation.

information retrieval (IR), machine learning (ML), rule-based (RB) and agent-based (AB) techniques to provide personally adapted support “behind” one’s favorite email system. We view the combination of IR, ML, RB and AB techniques using powerful open source components as a key enabler to providing a robust system.

Like our vacation agent, we can leverage agent interaction, information sources, and combine IR, ML and RB to attain more “intelligent” and useful agent behavior.

Future tasks we envision include:

- Summary and digest (especially when lots of email arrives about a topic)
- Apply (e.g, use email to trigger and run specified actions)
- Notify (e.g., send a summary or alert about selected email to a users pager or voicemail).
- Smart Vacation which responds to people with a tailored response, for example, co-workers may receive alternate contact information, known contacts may receive a return date, but unknown senders merely receive the information that the user is on vacation.
- Meeting Minder which monitors incoming email for messages about scheduled meetings. This agent may collate all messages, discussions and documents relevant to the meeting and make this information available to the user at the time of the meeting.

As we will detail further, the key contributions of our work are in the specific and rich features provided to handle email, in the use of interacting agents (for example the vacation agent), in the way in which the tools, agents and assistants work “behind the scene” to augment the experience using the user’s favorite email tools (e.g., Outlook and Exchange), and in the “tasteful” and extensible combination of IR, ML, RB and AB techniques using powerful open source components. We use the term "assistant" to suggest the overall style or flavor of the application - the PEA or PA "assists" a user in performing complex or rote tasks, such scheduling meetings or manageing email. The term "agent" refers to a technology and architecture, in which applications are partitioned into autonomous, loosely coupled components that discover and communicate with each other by exchanging highly structured messages.

In the remainder of this paper we start with a summary of background on personal assistants and related email work (section 2), describe our smart vacation multi-agent prototype (section 3), and then describe the design of our current email handling components (section 4). We conclude with a discussion of the status and experience of the current system (section 5) and planned and proposed next steps (section 6).

2 Background

In this section, we describe the CoolAgent personal assistant we have prototyped and some related email-handling systems.

2.1 The CoolAgent personal assistant

The CoolAgent personal assistant (PA) is an autonomous, proactive agent-based tool, implemented as a society of collaborating agents that support an individual user, and interact with group services and personal assistants of other users.

---

[2] These elements can be separate agents in some cases, or behaviours and loosely coupled classes in others.

The primary PA application demonstrated so far is that of distributed meeting scheduling [Griss et al, 2002a], as well as the new email agents described in this paper. As described, the user's PA is customized by a preferences and profile file containing personal information and preference rules about notification modes, meeting times, participants, and rooms. The PA interacts with the user's calendar agent (which can interface to one or more of several calendaring systems, such as Outlook/Exchange and Palm desktop), and the voicemail system. It can also interact with system agents (such as a meeting arranger agent) and services (such as a teleconference reservation system); the meeting agent can interact with other personal assistants and room reservation systems.

The PA is implemented as a set of multiple, collaborating JADE agents [JADE, 2000]. JADE is a Java open source system that is FIPA compliant. JADE uses a standard Agent Communication Language (ACL) to communicate between agents, and supports agents distributed on multiple machines, as well as mobile agents. We have extended the JADE system to produce a robust platform called BlueJADE[Cowan et al, 2002] by combining JADE and the HP AS J2EE application server[3], and by using UML hierarchical state machines to more precisely and flexibly specify behaviors [Griss et al, 2002].

The PA can route messages and notifications via one or more channels (email, voicemail, IM/jabber, or pager), based on preferences and context. Subsequent work also interfaced the PA to the CoolTown web-presence manager[CoolTown], allowing some location context and events to further adjust how the PA would respond to requests for meetings, and presence in a meeting room.

One part of the overall PA vision is for the PEA to interact as a peer or child of the other agents. In one direction, the PEA uses the calendar agents, the notification agents and the PA to find information and to communicate with the user. In the other direction, email messages concerning meetings could trigger the meeting agents, or at least monitor, prioritize and route email relevant to specific meetings.

2.2 Related email agent work

The notions of using intelligent agents, machine learning and collaborative applications to handle email and other personal information for individuals and teams are not new. However, many of the ideas described in earlier work are fragmentary, or only demonstrations. In part, this paper highlights how we have implemented a rich combination of these ideas in order to make a practical, extensible system that we can use in our daily work. In part, our contributions are only now possible because of the recent availability of a number of relatively high-quality, open source building blocks that enable us to use best-in-class capabilities, rather than the very simple, fragmentary elements heretofore feasible.

In any event, the large number of papers, systems and experiments show that this continues to be a rich area for experimentation with powerful techniques.

[3] An open-source version based on the JBOSS open-source application server is in final stages of preparation for release.

2.2.1 Widely used (commercial) email filtering systems

Many email clients and systems provide some kind of junk mail or SPAM filter and also sometimes a more general set of email handling rules. The large number of such systems testifies to the importance of providing assistance to the user in dealing with the deluge of email. For example Hotmail, Netscape, and Outlook each provide a way to define email filtering rules, or at least junk mail filtering rules. Some require the user to write rules; others use some kind of learning algorithms, or extraction of patterns from examples.

As a detailed example, **Microsoft Outlook and Exchange** both offer (essentially the same) way of writing rules that are applied to each incoming or outgoing email message, or can be run manually over any folder. The Office 2000 "rules wizard" can create some 11 types of rules that run on either the server or in the client. Each rule has a condition and an action. Rules are checked in a specified, user controlled order. Some rules run when a message arrives, others after it is sent, and others assign or modify categories based on content or header information. The set of conditions includes checking To, From and/or CC: lists, the presence of specific words in the subject or body, presence of attachments, etc. The actions include refiling, deleting, and forwarding a message, setting attributes and categories, printing the message, playing a specific sound or activating a specific application. Subsequently, the Outlook display can sort or filter on these attributes and categories. Complete sets of rules can be imported. Clicking on a message and using it as an example to construct a rule can invoke a different "organize folder" wizard. An "out of office assistant" provides a simpler/different interface for constructing and activating some rules (which may be a subset of the full rule system). Finally, junk mail and adult content filters are provided as a pre-built set of rules that look for specific structure, addresses or words in the sender, subject or body. Because only specific words are searched for (rather than invoking a user-specified or learned pattern), rules can become quite complex. Finally one cannot access other information to make filtering decisions such as forwarding or replying to email that arrives when certain information is in the calendar.

Outlook/Exchange also provides a tool to search for mail, notes, tasks or attachments of interest, using simple queries that look for words, presence of attachments or attributes, and size or date constraints. Search is slow because each message is scanned at search time rather than using an inverted index to locate matching messages directly.

For comparison, **MSN HotMail** provides a somewhat simpler Mail Handling capability. It has a separate *Junk Mail Filter* that keeps unwanted e-mail from reaching your Inbox. Its level of rejection can be set high or low, and messages caught in the *Junk Mail* folder can be marked as 'not junk' to lightly "train" the filter. Optional *Immediate* or *Delayed Junk Mail Deletion* can be selected; HotMail includes a *Safe List* of names and *Mailing Lists* from which messages will never be filtered as 'junk,' and an ability to block the delivery of e-mail from specific addresses. It also provides the ability to create up to 11 ordered *Custom Filters* that refile incoming messages to specific folders if specific words or phrases appear, do not appear or start or end the To, From CC, or Subject fields. Finally, one can select *Alerts* that arrange for you to receive notification on your mobile device when new e-mail arrives (via MSN mobile), but there do not appear to be any rules to determine which messages trigger notification.

2.2.2 Machine learning and Information analysis techniques

ifile is a general [ifile] email filter that adaptively filters email based on previous user actions and some preferences. ifile works with a mail client to intelligently filter mail according to the way the user tends to organize mail. ifile uses the machine learning algorithm Naive Bayes to classify e-mail documents. ifile is different from other mail filtering programs in three major ways:

1. ifile does not require you to generate a set of rules in order to successfully filter mail
2. ifile uses the entire content of messages for filtering purposes
3. ifile learns as you move incorrectly filtered messages to new mailboxes

See also the Stanford junk mail filter[Sahami 1988] which also uses Bayesian techniques to build an email filtering tree. The idea is to use adaptive, machine learning techniques to avoid the laborious and error-prone manual construction and maintenance of a set of filtering rules.

Such intelligent and adaptive filters are a critical part of the mail assistant. We are exploring the possibility of incorporating such modules into our system. It remains to be seen whether **ifile**, for example, is modular enough to be accessed from our Java code. Certainly the existing package, using C and Perl scripts, would require considerable effort to bring into our development environment. In addition, using an existing module restricts the choice of machine learning algorithm, e.g. **ifile** uses only Naïve Bayes.

SpamAssassin is a PERL-based tool targeted at Unix email users. It uses several rule-based techniques and heuristics to identify SPAM; rules and criteria are loadable or customizable from text files, so can be easily modified by users or administrators. Techniques used include:

- **header analysis**: spammers use a number of tricks to mask their identities, fool you into thinking they've sent a valid mail, or fool you into thinking you must have subscribed at some stage. SpamAssassin tries to spot these.
- **text analysis**: again, spam mails often have a characteristic style (to put it politely), and some characteristic disclaimers and CYA text. SpamAssassin can spot these, too.
- **blacklists**: SpamAssassin supports many useful existing blacklists, such as mail-abuse.org, ordb.org or others.
- **Razor**: uses Vipuls razor collaborative spam database to filter widely distributed SPAM messages. (*See below*)

Other systems include Netscape mail handler and Unix mutt/procmail.

2.2.3 Collaborative filtering

Vipul's Razor is a distributed, collaborative, spam detection and filtering network[Vipul, 2002]. Since spam typically operates by sending an identical message to hundreds of people, Razor short-circuits this by allowing the first person to receive a spam to add it to the database -- at which point everyone else will automatically

block it. Through user contribution, Razor establishes a distributed and constantly updating catalogue of spam in propagation that is consulted by email clients to filter out known spam. Detection is done with statistical and randomized signatures that efficiently spot mutating spam content. User input is validated through reputation assignments based on consensus on report and revoke assertions that in turn are used for computing confidence values associated with individual signatures.

2.2.4 Agent-based systems

Information Lens system [Malone et al., 1987] was a pioneering groupware tool in which intelligent agents help users find, filter, and sort large volumes of electronic information. The Lens system was written in object-oriented Interlisp-D. It has a central server named "Anyone" which receives messages that include "Anyone" as an addressee from the existing mail server. By automatically sorting and periodically retrieving messages from the special mailbox, "Anyone" sends the message to several additional recipients whose rules select it. The Lens system allows users to build rules for finding, filtering, and sorting messages. Rules consist of a test and an action. If a message satisfies the test, then the action specified by the rule is performed on the message. As in composing messages, the system also provides a display-oriented editor for constructing rules by filling the fields of a rule template. This template-based graphical rule construction was found to be very easy for inexperienced computer users. The same approach to rule construction is used in the Outlook/Exchange system. See also GroupLens [GroupLens].

Patty Maes [Maes, 1994] describes a vision of a set of agents ("Agents that reduce work and information overload") that support a single user for several tasks (such as meetings, email,…). While she talks about agents collaborating with other task and user agents to improve performance, it appears that not much was done in this area. She also described some initial experiments.

POSTMAN [Postman, 1999] is a personal e-mail filtering agent that can help users to classify incoming e-mails according to the rules input by the user. This system is similar to the Information Lens, and to Outlook, in that users need to edit processing rules by hand. However, it is more practical than the Information Lens, being built on the PINE email system and is written in JAVA rather than LISP.

Intelligent Email Agent [Florea and Moldovanu, 2001] is an email handling agent that helps a UNIX mail user. It has a simple learning, model but does not use body keywords, nor does it execute any reply action. It supports Boolean AND/OR patterns using a restricted set of predefined key fields from messages (using the attributes: *subject, from, CC, date, length*). It uses Quinlan ID3 decision tree learning algorithm (see WEKA). The Florea and Moldovanu paper contains a good discussion of some of the learning issues with email. See also [Lucene]'s Boolean, weight and affinity capabilities.

In summary, there are a number of systems that provide some useful amount of email handling assistance, through a selection of one or more pre-built or user-defined rule sets, through some adaptive (Bayesian) techniques, or through some collaborative or agent-based techniques. Many of these have a rather rigid notion of what the user can customize, or how the mail handling interacts with other applications and information sources.

In our work we wanted to build a more flexible and robust framework that allowed us to combine several powerful, high-quality, implementations of information retrieval, machine learning, rule-based and agent-based (collaborative) techniques to produced a highly configurable system, with more features than found in a typical system today.Our ideas on how to structure the system have been influenced by the venerable RAND Mail Handling (MH) system [MH 1992; Peek 1995]. The MH system is notable as a set of Unix components and filters (a "kit"), rather than a monolithic mail system. Each component handles a specific mail task, such as deleting, refiling, sorting or folder management. It is fairly easy to link the components into various mail processing tools using shell scripts, perl scripts, or frontend email readers such emacs (via mh-e), xmh, exmh or vim. Recently, the MH book has been updated to include nmh, and is also available online[4].

3 Initial Experiments: Email Vacation and Search Agents

In this section, we describe our initial experiments in applying an early set of email components and a suite of agents to the email portion of the personal information space. These experiments led us to define and implement a new set of more powerful mail handling components, described in the next section. When these are all completed, we will then enrich these email agents, and also expand to other information sources and modes.

We developed a simple vacation agent and email search agent, as well as several supporting agents. We use several new agents that interact with some of the agents previously developed for the PA meeting assistant. These new agents include a "mail agent" that receives and routes incoming email, a "mail indexing agent" (using Lucene to build inverted indices) to support a fast "mail search agent," a "contacts agent" and a "vacation agent.." Lucene is an information retrieval/indexing package[Lucene] and WEKA is a package of machine learning algorithms[WEKA]. Both are described described in greater detail in section 4, and used extensively in our work.

The email agents are supported by email hooks that access incoming mail. We have developed two kinds of appropriate email hooks to allow intercepting, (re-)routing, preliminary analysis and categorization of mail. The first uses the Javamail API. The second is a new Java Exchange Bridge (JEB), written by us to provide finer-grained access to the Exchange services, using the JACOB open source toolkit [JACOB]. JEB enables us to access Exchange services that are not available via the Javamail API, such as contacts and calendar data. It also exposes more message structure particular to the Exchange server, which we use, for example, to insert a priority field in a message. The bulk of message processing in the email agent, however, uses Javamail.

The two supporting agents used by the email agent are the calendar agent and the contacts agent. The calendar agent may access the Exchange calendar service using VIEW, an HP-written web-service that provides *https* access to Exchange, or JEB. The contacts agent accesses stored information about "me and my associates." Initially it stored contacts data in an XML file; now it can access contacts data on Exchange using JEB.

[4] Unfortunately, nmh does not run on the Microsoft Windows operating system, and so could not be used as part of the solution; perhaps it can be used for a future Unix/Linux version.

The vacation agent uses both the contact information and the calendar information to decide if and how to respond to email messages sent to me. It first finds information about "me" from the contacts agent. It determines whether I am on vacation by periodically asking the PA. The PA uses several heuristics, and may query the calendar for information about my scheduled vacations, or other scheduled unavailable times. For each incoming message, the vacation agent checks to see if a response has already been sent to the sender, using a mySQL database to store contacts to whom it has previously responded. If not, it constructs and sends an appropriate email response. Using contact information the vacation agent recognizes a person even when messages arrive from different email addresses. For example, ruth_bergman@hp.com is the same person as ruth@hpl.hp.com. Last, the vacation agent uses some additional heuristics to avoid responding to bulk mail messages, such as mailing lists.

The key advantage of the vacation agent over existing vacation response capabilities in mail clients is that the user does not have to set up the behavior of the client upon leaving the office. Rather, the appropriate behavior will be triggered, by finding the vacation event in the calendar. So the user can create the vacation event months in advance, while planning the vacation, with no further action required. For last minute vacations, assuming the user has some access to his calendar, the user may add a vacation event at any time and trigger the vacation response behavior.

These initial experiments helped us understand how to use Lucene effectively, the difficulty of managing large amounts of email flowing as ACL messages, and issues related to synchronizing the inverted index and the processed mail. These lessons led to the new mail event driven architecture discussed below. The next step of additional email filtering, similar to an *ifile*-like [ifile] email filter that adaptively filters email based on previous user actions, some preferences, and additional profile information. Using this new architecture and components, we also plan to create additional services and agents, such as a SONIA-like [Sonia] meta-search engine and a bookmark organizer.

4 Architecture and Design

In this section, we describe the set of email handling components, the notion of a flexible dispatch tree, and how these components can be used with or without the agent system. Following our initial work on extending the PA to handle email, we have designed and are implementing an even more powerful and flexible set of email handling components and events, which can be used both standalone or within the agent context.

4.1 Flexibility

We want to be able at configuration time or dynamically to add new kinds of email "filters," modify preferences, and add new learned rules and mail classifiers.

We want to combine these email "filters" into a standalone personal email management system, as well as use them as key components in an individual or (collaborative) team-oriented personal mail and meeting assistant. For example, the "vacation agent," "calendar agent," "meeting agent" and "contact agent" should be able to interact appropriately, with each other if present, and with corresponding agents of other team members, as appropriate.

In the CoolAgent system, we developed a property-file customized event dispatcher, with "loadable" Activities[griss et al, 2002]. ACL message events, timeouts and other system exceptions are converted into subclasses of Event, and uniformly distributed to Activities by a set of ordered Dispatchers. Some of these Activities and Dispatchers can themselves be Dispatchers (producing a dispatch tree) or State machines if more precise event ordering is required. Dispatchers and Activities include an EventTemplate pattern that matches incoming Events to decide if this dispatcher should handle this event. If we treat email as an Event, we can use a Dispatcher-like object as an event filter, which selects an appropriate Activity to handle the mail event. The property-file can then be used to configure a mail handling dispatch tree.

4.2 The basic classes

We wanted our eMail handling components (*MailHandler* or "mail filters") to be as compatible with the CoolAgent event-driven model as possible, yet still be completely useable as an independent mail handling application.

The basic idea is to (statically or dynamically) build a mail filtering/workflow-like tree using configuration properties[5]. An instance of this tree is created and activated for each incoming email message or event.[6] As the email message or event flows through the tree, it is annotated and actions are invoked, as appropriate.

4.2.1 Mail events and property flow

Incoming email messages and other email related events, such as delete message, create folder, etc., are converted into *MailEvents*; MailEvents are subclasses of PropertiesEvent, a subclass of Event that includes a BasicProperties object. The BasicProperties object is a list of name-value pairs, and allows each MailHandler to annotate the incoming mail event to indicate what processing has been done, and add other useful information that a later MailHandler might use in its processing.

4.2.2 MailHandler components

All mail handling components implement the same *MailHandler* interface, which includes a MailEvent handling method for all possible mail events. These events include: messageAdded, messageRemoved, messageChanged, folderCreated, folderDeleted and folderRenamed[7].

At initiation, the top-level component, called MailTool, constructs the tree of MailHandlers, instantiating each, using a loadMailHandlers method.

---

[5] This approach nicely merges the capabilities of CoolAgents' Dispatcher and HSM, which allows these components to be used later when we integrate with agents. We have chosen to describe and implement MailEvent as a true subclass of CoolAgent's Event or PropertiesEvent, and MailHanlder as a true subclass of Dispatcher (both well described in [Griss et al, 2002]).

[6] If we are not using threads or asynchronously communicating agents, we only have to construct a single instance of the tree at startup time, and incoming mail events are processed sequentially. We believe that individual email events can be processed rather quickly, but that each time the overall mail handling system is started, there is a significant startup cost to fetch the contacts db, etc.

[7] Other events, such as messageMoved, folderMoved, messageForwarded, messageRepliedTo, are not yet fullyhandled.

Upon receiving mail messages or events, MailTool converts these events into MailEvents, and calls the appropriate event handling method of each mail handler in the tree, in depth first order. The event handling method returns a flag indicating that further processing on this mail event should be cancelled. If this flag is set no further processing is carried out on that branch of the tree.

Each MailHandler is free to test values and properties of its incoming MailEvent, and return immediately. As each Mailhandler performs processing of the MailEvent, it can change and add new properties, to communicate with its children or parent mail handlers.

Each MailHandler can be passed a set of command line properties, including an "*import:file*" to customize it in some way as it is instantiated in the tree. For example, the rule-based RuleHandler component will pass in the name of a rule file.

4.2.3 Standard MailHandlers

Components we have implemented or are implementing include a TopLevel MailTool component ("root"), called by the system, to convert all incoming mail, mail events, and other messages into appropriate subclasses of MailEvent.

Note thatthe system (or some special MailHandler components) will maintain synchronization information, so that if some part of the system is unavailable, mail will be processed the next time.

For each new message, the components applied include:

- Contacts – Consult the Contacts database (using JEB, or local cache), and add properties to distinguish and appropriately deal with well-known contacts.
- Classify – Uses extracted email properties and Contacts annotations as input to a classification tree (see WEKA) setting the additional properties that will trigger subsequent MailHandlers or rules.
- Index – Extracts mail keywords, and maintains the set of inverted Indexes (using Lucene).

Other MH components are used to react to email events such as delete or refile, or to request those actions. For deleted messages, it deletes the message, or moves it to a deleted folder.

- Refile – if appropriate, moves the message to another folder for later processing in that folder.
- Vacation – checks the calendar (or a periodically refreshed context object) to see if an appropriate version of an "on vacation" or other "status" email corresponding to this sender.
- Junkfilter – if appropriate, decides to remove this mail from further processing, either deleting it, or filing it in a Junkfolder.
- Digest – adds a summary of the message to a file or database entry, that will subsequently be turned into a digest message for later processing.
- Prioritize – computes a combined mail priority based on presence, absence or value of other fields and properties, and also sets this value in the Exchange copy of the message so that Outlook can (optionally) display the messages in this order.

So for example, we might use the following simple configuration file to define a three-level processing tree:

```
root.count=2
root.1=junk:com.hp.mh.JunkMailHandler
root.2=vacation:com.hp.mh.VacationHandler

junk.count=2
junk.1=myjunk:com.hp.mh.FromFilterHandler(from:me)
junk.2=yourjunk:com.hp.mh.FromFilterHandler(from:you)

vacation.count=3
vacation.1=contacts:com.hp.mh.ContactHandler
vacation.2=check:com.hp.mh.RuleHandler(rules:vacation-rules.jess)
vacation.3=respond:com.hp.mh.SendHandler

respond.count=3
respond.1=forward:com.hp.mh.SendHandler
respond.2=refile:com.hp.mh.RefileHandler
respond.3=delete:com.hp.mh.DeleteHandler
```

4.3 Information Retrieval and Machine Learning

We use the open source systems Lucene [Lucene] for information retrieval (IR) capabilities and WEKA[Weka] for machine learning (ML).

IR techniques are used to extract key features from email elements as "structured documents." In addition, a good IR system will support topic discovery, word spotting, and lexical affinities. It should support multiple document types (email, web pages, and files) and simplify search and classification by building inverted indexes. Some of the inverted index construction ("Indexing") can also be used to collect frequency and term correlation information of relevance to the machine learning system.

For our ML system, we wanted a system that would allow us to represent and learn individual and team preferences, learn topic hierarchies, and support manual and learned information classification.

See also the discussion in [Sebastiani 2002].

4.3.1 Lucene

Jakarta Lucene [Lucene] is a high-performance, full-featured text search engine written entirely in Java. It is being used in the PEA to build inverted indices of all the email for an individual user. Lucene provides fielded data and allows fast queries of the form '"title:"a paper" AND text:go,' with various levels of wild card, fuzzy, proximity and priority search.

As mail messages (or other documents, such as attachments) are processed to create the set of index terms under which the document is indexed, various stoplist and stemming word and phrase filters can be used to make the set of index terms more uniform and more useful for finding, analyzing and annotating mail messages. Also, dictionary files can be used to handle aliases, etc. The same analysis and filtering are applied to search queries to improve the search precision.

Furthermore, Lucene provides elaborate control over the information stored in the index for each document and how this information is used during indexing and searching. On one extreme, you can store for each document just its location (e.g. URL) and index the content of the document as a monolithic piece of text. On the other extreme, you can store the entire document as well as various attributes such as Author, Title, and Date and perform searches that consider these attributes for matching and ranking.

When we run Lucene over our mail files, it creates a set of index files locally, allowing for very fast search. The local Lucene files also serve to indicate which messages have been indexed, and so also keep track of synchronization information. There are various ways we can use the indexing phase to support subsequent steps in the MailHandler tree. For example, we can determine important term frequency information, which could be relevant to generating more useful classifiers. Since the indexing process can extract or generate a canonical set of keywords/key-phrases to tag each message (stemming and aliasing), classifiers can run over these terms, rather than, or in addition to, the raw messages. Also, we can access and use the terms that index a particular message to support fast canonicalized queries to test if this message is "about 'xxx'. "

4.3.2 WEKA

WEKA [WEKA] incorporates several standard ML techniques into a software "workbench" called WEKA, for Waikato Environment for Knowledge Analysis. With it, a specialist in a particular field is able to use ML to derive useful knowledge from databases that are far too large to be analyzed by hand. WEKA's users are ML researchers and industrial scientists. WEKA is open source software issued under the GNU General Public License. WEKA provides both a machine learning framework, and a collection of machine learning algorithms for solving real-world data mining problems. It is written in Java and runs on almost any platform. The algorithms can either be applied directly to a dataset or called from your own Java code. WEKA is also well suited for developing new machine learning schemes. The WEKA distribution contains implementations of many common classification and regression schemes. We use the WEKA support vector implementation of John C. Platt's sequential minimal optimization algorithm [Platt 1998] to learn the junk mail filter. In addition we implemented, in the WEKA framework, a rule-based classifier that is not learned. This type of classifier is used for some of the mail event classifications, such as deciding whether a mail message is addressed to "me,," the user.

4.3.3 Support vector machine - LIBSVM

We also will use a support vector package (LIBSVM), [Chang and Lin, 2002]. It provides a powerful automated classification capability, well matched to the needs of text classification used in the PEA and other information assistants. We are considering adapting LIBSVM to fit within the WEKA framework. While WEKA provides a support vector classifier in the SMO class, LIBSVM is widely used and provides a new ν-svm package [Schölkopf et al 2000]. Unlike the standard C-SVM formulation for support vector machines [Vapnik 1998], which penalizes the total misclassification error, the ν-SVM formulation lets one control the number of support

vectors and errors. The ν parameter is more intuitive to tune than the C parameter, which is difficult to select.

4.4 Machine Learning Applied to Email

Incoming email is classified into a rich set of general categories according to learned model and some preferences. Then these classifications are input to the action engine driven by rules, preferences, dispatch trees, state machines, etc. and context to select appropriate actions.

The current implementation uses several hand-coded classifiers to do a initial partitioning of the email. The next step includes the use of one or several learning strategies to generate and refine learned classifiers. These strategies include:

- Supervised - based on analysis of a log of prior actions corresponding to observed disposition of email in folders, from a log of the user's actions, or from advice solicited from the user.
- Unsupervised – Some automated clustering or related techniques can partition the email, e.g., discovery of a new discussion topic, new mailing list or new meeting
- Reinforcement – based on periodic user feedback on the effectiveness of the current classification and disposition, the system might adjust the classifiers, or other parameters to better satisfy the user.

As the user gains confidence in the recommendations or actions made by the system, the user might increase the autonomy delegated to the system; initially, the system might start very conservatively, and ask the user for guidance, later it could do more things automatically. In general, we do not let the PEA permanently delete anything; email is moved to junk or deleted folders (perhaps even this can be learned).

Furthermore, we imagine some learning done in "batch" or "offline" mode, some "online" with continuous correction by the user, and some "collaborative," in which our PEA can consult the PEAs of other users.

In our implementation, each MailEvent is processed by a ClassificationMailHandler, which submits the MailEvent to each of the classifiers in a bank of classifiers. Each classifier sets a property in the MailEvent to either True/False (boolean classifier) or to some percentage or confidence level (numeric Classifier). Subsequent mail handlers, then, have access to these classifications.

4.5 Combining Machine Learning and Hand-coded Rules

After the classification step, each rule enabled MailHandler, (a subclass of MailRuleHandler) will apply rules to the annotated values and property sets, including the properties created by the classifier. When a rule matches, it will invoke the corresponding Actions in the MailHandler. These rules and actions can be coded directly in Java, but can also be written in a specific rule language, such as JESS™, or the the CoolAgent Matcher, a structure pattern matcher which is under development as part of the CoolAgent toolkit[Griss et al., 2002a], or a specialized XML rule form with user-friendly rule editor (See also Outlook rules.)

For rules, the PA currently uses JESS™, with extensions to support fuzzy sets and fuzzy logic. JESS is a forward chaining system (based on CLIPS),which uses

reflection to integrate well with Java programs.[8] It is used to represent some preferences and policies in the PA.

Each MailRuleHandler will specify a rule file which contains a ruleset (an ordered list of rules, each rule consisting of a conditionPart and an actionPart; the order determines which rules will be tried first) This ruleset will be loaded when the mailRuleHandler is instantiated; it may optionally be compiled (e.g., Jess forward chaining rules). [9]

For our initial experiments, we have implemented three kinds of RuleHandlers: some are directly coded in Java, some are implemented using a Jess-based RuleHandler, and some are built from a simple XML mail-list filtering rule set.

For example, the following is a fragment of the mail list filter/refiler:

```
<ruleset>
<rule>
  <folder>Email/Administrivia/financial</folder>
  <from>SchwabAlerts.MyPortfolio@Schwab.com</from>
</rule>
<rule>
  <folder>Email/Internet/com</folder>
  <or>
        <from>advantage@mac-mall.com</from>
        <from>advantage@pc-mall.com</from>
        <from>specialoffers@specialoffers.onvia.com</from>
  </or>
</rule>
</ruleset>
```

The structure of Jess files is more complex than just an ordered list of rules; rules may be assigned a weight (called “salience”), and auxiliary functions can be defined as predicates for the conditionPart, or actions in the actionPart.

Several RuleHandler components can appear in the tree, each with its own rule file.

4.6 Presentation of Processed Email

Email can be presented visually using Outlook, or some other mail reader. In Outlook/Exchange, it is possible to tag email with additional user fields, which can then be used in some Outlook views to prioritize or filter the mail.

In the agent version, some mail handlers may invoke an agent with an appropriate extract from the message; for example, selected email can send an ACL message to the Notification agent that is part of the PA; this agent can appropriately forward an alert to the users voicemail, pager or phone, or use other preferences rules to determine disposition.

---

[8] It can also handle some forms of backward chaining.

[9] There are several efforts to standardize a Java API for a simple rule system (JSR-94)[JSR94], and also some work on the standardization of a rule language in XML called SRML[SRML] that will be compatible with JSR-94. These will influence the next iteration of our system.

4.7 Profile and Preferences.

In the PA, preferences are represented as a mixture of properties files, XML files, JESS rules and RDF/semantic web models. As we further develop the classifier/rule integration, we will need to make this representation more uniform, and address the issue of layered preferences, context-based changes in preference sets and individual vs. team information preferences. This has a significant impact on preference learning, analysis, and composition.

5 Status

5.1 Experience

At the time of writing neither the MailAgent nor the MailTool have been released for public use. We can, therefore, report only on our own experience using the tool. Naturally, having developed these mail assistance tools, we have built in those capabilities we most desired. Among our team members, each new capability has been received with cheers, and the existing set of capabilities in the MailTool is sufficiently complete that we are now running the tool continually on our desk top machines.

Each member of our team has a different mode of email use, and, indeed, prefers a different subset of the capabilities we have described in the paper. The MailTool's design using loadable mail handlers enables the user to turn functionality on and off easily. We have found this flexibility very useful, and each of us has tailored the MailTool to our individual preferences.

Email systems are very dynamic in nature, and robustness has been an issue during development. More testing is necessary, including using the tool continuously for long periods and by a larger group of users. We have found that each user, with his or her mode of use, tests different aspects of the system. Overall, we feel that the current functionality is sufficient to warrant an alpha release of the tool in the near future.

5.2 Mail Issues

We have found email to be a challenging environment. The mail delivery system is extremely dynamic. Mail arrives irregularly. The system may be idle for long periods and extremely overloaded at others. In addition to the arriving mail, there may be several mail clients modifying the mailbox. Somewhat absurdly, from the perspective of a mail handler, the user is very problematic and can sometimes pull the rug from under our feet. For example, a mail handler may be processing some message and the user can come along and delete it mid-processing.

A comprehensive exception handling and failure recovery system should be put in place. The mail agents need to seamlessly adjust to messages appearing, disappearing and moving. Likewise, folders may be created, deleted and renamed by the user or other clients. Last, the mail server or any other part (agent) of the system may fail to respond. We need to make the system robust in the face of all such situations or failures, by having "caches" of hints, ensuring that synchronization restarts appropriately.

During event processing, if the MailHandler tree takes a significant chunk of time to complete it's processing, the whole system could bog down. We would then consider using MailEvent queues and several threads (perhaps via a thread pool), each running its own instance of a processing tree.

It turns out that the IMAP protocol used by Javamail, and the MAPI protocol used by JEB have several incompatibilities. Since the mail tool uses both protocols, there have been some challenges in making them work together. We have been forced, at times, to delve into the details of these protocols in order to understand the strange behavior of the system.

The main idea is to design so that the mail agents largely provide non-destructive assistance to the user, with the final decision left to the user. At worst, the user will then be no worse off than if little or no assistance were provided.

5.3 Agent Issues

When embedding the mail handlers in an agent, the system becomes even more dynamic. In addition to mail events, ACL messages come and go in the system. The event-handling scenario is far more complicated, and it is difficult to ensure robustness. In the agent environment, we need superb exception handling and failure recovery.

An agent that handles mail must be able to communicate about mail. Thus, an appropriate ontology is required. It is not difficult to create a mail ontology, in particular, since mail protocols exist and within them the contents and meta-data for mail messages are defined in detail. Contact information, on the other hand, is more difficult to define. An ontology that is too simplistic will not support the needs of the users or agents, whereas an overly detailed representation becomes cumbersome. For any user, most contacts can be represented by a very simple ontology, but there will be several contacts for which that representation does not suffice. Some information about a contact may be useful for a user, but irrelevant for the agent. The opposite may also be true. For example, a user only needs one current email address for a contact. For an agent to recognize that two mail messages are coming from the same person, all possible sending addresses must be known.

While we can say that the ontology for mail messages is already defined by mail protocols, the size of messages, becomes an issue in an agent environment. Agents may transport these messages multiple times in the agent communication system. This problem is exacerbated by large messages or messages with attachments, which can be very large. Rather than including all the text of the email in the ACL message, one solution to this problem is to embed a reference to the email in ACL messages, for example, using the email index to point to the email text as a document, and to the optional attachement.

Last, all the information required to connect to the mail delivery system and rules and preferences that govern the mail handlers must be embedded in the agent's profile. Although we have designed this system to handle mail behind the scenes, some capabilities must have a user interface, e.g., the mail search function. When an agent has access to the user's mail, it opens up a channel of communication between the user and the agent. The user may make requests from the agent via email. The agent can also inform the user via email. We have not taken advantage of this human-agent interface yet, but view it as an opportunity with great potential.

6 Conclusions and Next steps

In this section we summarize our accomplishments and contributions, and hint at next steps.

6.1 Key accomplishments and contributions

We have built a powerful system that looks like members of our team in their daily work can use it. It is robust and feature-full enough. It can be used standalone, or with the agent-based PA. In the agent form, it does not depend on other users also running their email agents, nor does it require that the email agent be operational all the time.

The key contributions of our work are in the specific and rich features provided to handle (filter, refile, and prioritize) email, and the flexibility with which these features can be combined and coordinated.

An aspect of the design that we have only partially exploited, is the use of interacting agents (for example the vacation agent) and the interaction of the agent-form of the mail handler with our Personal Assistant agents.

A key design goal is that the tools, agents and assistants operate "behind the scenes" to augment the experience using the user's favorite email tools (e.g., Outlook and Exchange), by intercepting, modifying and reprioritizing mail displayed by the normal outlook/Exchange combination.

Finally, the "tasteful" and powerful combination of information retrieval, machine learning, rules and agents is enabled by the use of many high-quality open source packages, notably Lucene, WEKA, MySQL, and JADE. This allowed the development of a robust and practical solution.

In summary, the features we current support, or have designed for, include an integration of several elements:

- Monitoring the current email stream
- Accessing an archive of existing email
- Use of history of (recent) actions
- Initial and ongoing training to improve
- Handling of multiple email "actions": delete, copy, forward, delegate to agent, run though application, put in a folder (look at Outlook and SNOOP)
- Customizable level of autonomy, depending on trust, accuracy, preferences
- (A future) goal of immediate usability without much pre-customization (inherit default profile and rules, extract key categories from sample of recent email, etc.

We have designed for several use cases:

1. Learning process and interface:
    - Edit current rules
    - Setup initial rules
    - Browse action history and provide feedback
    - Monitor current actions and provide feedback
    - Define legal actions
    - (Re-)train on recent corpus

- Change level of dialog and confirmation, in general, and per confidence on classification
- Edit learning parameters, degree of autonomy

2. (Autonomous) Email Filtering Assistant
   - Identifies key messages related to tasks, goals, schedule, meetings
   - Automatic routing to delegates, roles
   - Summarizes for nomadic, slow access
3. (Smart) Email Responder and interaction with other PA agents
   - Composes appropriate responses and routings to incoming messages based on subject, current schedule, some content words, goals, etc. (e.g., smart vacation responder). Uses calendar, preferences, context, to decide what kind of response, to whom.
   - Another example is a "meeting minder" agent - Watches for mail about known meetings or from meeting participants; route. If possible, convert to ACL, forward to other agents.
4. Information Indexing Assistant
   - Finds relevant information for upcoming meetings in email, local files and web

6.2 More Interaction with the Personal Assistant

6.2.1 Expand the agent-based solution

The next step is to refine and expand the initial experiments so that the email components interact as subsidiary agents/services to the tasks being carried out by the overall personal assistant or other task-oriented agents, such as the meeting agent. A number of new issues crop up from the need to share information among multiple agents, such as privacy, security, and control.

We can imagine applying the PEA to aspects of meeting preparation, arrangement and execution, and intelligent notification and routing of relevant email. (E.g., email about meetings, use of email formats, such as iCal to invoke meeting tools, the automatic forwarding of meeting relevant mail to participants, automatic meeting reminders, etc.) The PEA will use services (notification, calendaring, context management, etc,) provided by the Meeting Arranger Personal Assistant to jointly provide task- and context- sensitive filtering and composition of relevant information

Some parts of the MailHandler tree can then be embedded within separate agents, and some agent-to-agent communication is permitted to get additional information, such as asking the personal assistant or calendar agent for information, rather than accessing the Calender directly. This agent-directed dialog can be encapsulated as a special MailHandler. Likewise, some actions, like Notification via a non-email channel (email2voicemail, pager, …) can be carried out using the context-driven Notification agent.

6.3 Collaborative filtering of junk mail, and team management of email based on team preferences

In the agent environment, the transition from a mail handling agent to a community of mail handling agents is seamless. Our PEA is already a community of agents that communicate about and manipulate email. A natural next step is to communicate with other PEA's about email. By allowing communication between email handlers, for example about junk mail, we can produce a system similar to that of Vipul' Razor for collaborative SPAM[Vipul]. Estabilishing such collaboration among meeting management agents can ensure that all parties arrive at a meeting equally prepared, with access to a complete set of supporting materials.

6.4 General personal information management

A SONIA-like [Sonia] meta-search engine would collect search results from various search engines, download the returned pages, and post-processs them to cluster the pages into categories, and then rank the pages within each category. It should also rank the categories, provide a short description of the category (perhaps as a list of keywords), and provide a means for iteratively searching with feedback. It might also store previous searches like *Copernic* [Copernic]. It might also provide a simple interface to allow users to add their own search engines that might provide search capabilities for local information or servers.

A bookmark organizer would take a (perhaps slightly organized) set of bookmarks or favorites and create a Yahoo-like topic hierarchy. It should also be able to automatically incorporate new pages from the browser's history in the hierarchy, and update the hierarchy as new pages are added. Optionally, pages should be able to belong to multiple relevant categories.

A meeting manager would combine calendar information about a meeting, with mail messages about the meeting, documents related to the meeting and supporting materials from the web. These materials would be collated from the time of meeting request through the actual meeting. Some documents may arrive by email as attachments, some will be found by the agent using data-mining techniques on the user's local data and the web. At any time before or during the meeting the user can access materials, review the topics, update information.

7 Acknowledgement

We have gratefully drawn from the ideas and software of several groups. These include the original developers of the CoolAgent Personal Assistant, and the developers of the various open source toolkits we have built on, including JADE, Lucene, and WEKA. We also thank Dick Cowan, Ed Katz, Robert Kessler, Reed Letsinger and Marie Vans for many useful discussions and support.

8 References

[CoolTown] http://www.cooltown.hp.com

[Copernic] Copernic, http://www.copernic.com

[Cowan et al, 2001] Dick Cowan, Martin Griss, Bernard Burg , **BlueJade – A service for managing software agents,** HPL Technical Report, HPL-2001-296, Nov 2001. Also in AAMAS 2002 - Workshop on Challenges in Open Agent Environments, Bologna, Italy, July 2002.

[Cowan et al, 2002] Dick Cowan, Martin Griss, Robert Kessler, Brian Remick, and Bernard Burg, **A Robust Environment for Agent Deployment,**
AAMAS 2002 - Workshop on Challenges in Open Agent Environments, Bologna, Italy, July 2002

[Florea & Moldovanu, 2001] AM Florea & A Moldovanu, **An Intelligent Email Agent**, "Politehnica" University of Bucharest.

[Griss et al, 2002] Martin L Griss, Steven Fonseca, Dick Cowan, Robert Kessler **, Extending the JADE Behavior Model for more Flexibility and Control**
HP Laboratories Report HPL 2002-298(R), July 2002, Also, AAMAS AOSE workshop, Bologna, Italy, July 2002.

[Griss et al, 2002a] Martin Griss, Reed Letsinger, Dick Cowan, Craig Sayers, Michael VanHilst, and Robert Kessler., **CoolAgent: Intelligent Digital Assistants for Mobile Professionals - Phase 1 Retrospective,** HP Laboratories Report HPL-2002-55(R), Jul 2002.

[GroupLens, 1994] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, John Riedl, **GroupLens: An Open Architecture for Collaborative Filtering of Netnews**, From Proceedings of ACM 1994 Conference on Computer Supported Cooperative Work, Chapel Hill, NC: Pages 175-186, 1994.

[ifile] ifile, http://www.ai.mit.edu/~jrennie/ifile

[JACOB] Java/Windows object bridgeJACOB is a JAVA-COM Bridge that allows you to call COM Automation components from Java. It uses JNI to make native calls into the COM and Win32 libraries. See http://danadler.com/jacob/

[JSR94] A proposed standard Java Rule Engine API, Java Community Process, JSR-94. http://www.jcp.org/jsr/detail/94.jsp.

[Lucene] Jakarta Lucene, an opensource Information Retrieval/Text Indexing package, – http://jakarta.apache.org .

[Maes, 1994] Patty. Maes, **Agents that reduce work and information overload**, CACM, 37(7), p.31-40, 1994.

[Malone et al, 1987] Tom W. Malone.; Grant, K. R.; Turbak, F. A.; Brobst, S. A.; and Cohien, M. D. Intelligent information-sharing systems.
Communications of the ACM 30:390-402, 1987.

[MH 1992] The RAND mail handling system, ftp://ftp.ics.uci.edu/pub/mh/, and especially the online MH book by Jerry Peek.

[Peek, 1995] Jerry Peek, **MH & xmh: Email for Users & Programmers**, ISBN 1-56592-093-7, O'Reilly & Associates, Inc. , 1995. In June, 1996, ORA made the book freely available under GNU GPL. See http://www.ics.uci.edu/~mh/book/.

[Platt, 1998] J. Platt (1998). **Fast Training of Support Vector Machines using Sequential Minimal Optimization.** Advances in Kernel Methods - Support Vector Learning, B. Schölkopf, C. Burges, and A. Smola, eds., MIT Press.

[Postman, 1999] Jiang Chen and Haiwei Ye , **POSTMAN - An EMAIL Filtering Agent**, University of Montreal, C.P. 6128, Succ. Centre-ville
Montreal, Quebec Canada H3C 3J7, chen@iro.umontreal.ca
ye@iro.umontrea.ca, 1999.

[Schölkopf et al, 2000] Schölkopf, A. Smola, R. Williamson, and P. L. Bartlett. **New support vector algorithms**. Neural Computation, 12, 2000, 1207-1245.

[Sebastiani, 2002] F. Sebastiani, **Machine learning in automated text categorization**, ACM Computing Surveys, 34(1), March 2002, pp. 1-47.

[SELECT, 1999] **SELECT: Social and Collaborative Filtering of Web Documents and News**, The SELECT Project Team**,** In Proceedings of the 5th ERCIM Workshop on User Interfaces for All: User-Tailored Information Environments, Kobsa, A. and Stephanidis, C. (Eds.), Dagstuhl, Germany, Nov. 28th - Dec. 1st 1999, 23-37.

ERCIM (The European Research Consortium for Informatics and Mathematics). (Also as Technical Report CSEG/9/99, Computing Department, Lancaster University).

[Shamai et al., 1998] Sahami, M., Dumais, S., Heckerman, D., and Horvitz, E. **A Bayesian Approach to Filtering Junk E-Mail**. In Learning for Text Categorization: Papers from the 1998 Workshop. AAAI Technical Report WS-98-05.

[SONIA] http://robotics.stanford.edu/users/sahami/SONIA/SONIAproject.html

[SpamAssassin] http://www.spamassassin.org

[SRML] Simple rule markup language in XML, compatible with JSR-94. See http://xml.coverpages.org/srml.html.

[StaelinChang and Lin, 2002] Chih-Chung Chang and Chih-Jen Lin , **LIBSVM -- A Library for Support Vector Machines,**. See http://www.csie.ntu.edu.tw/~cjlin/libsvm/

[Vapnik, 1998] Vapnik, V., , **Statistical Learning Theory**. New York, NY. John Wiley, 1988.

[Vipul] Vipul Pred Prakash, An opensource "artistic licence" at http://sourceforge.net/projects/razor/

[WEKA] The University of Waikato, New Zealand, Machine Learning Project http://www.cs.waikato.ac.nz/~ml/