Meeting Runner:
An Automatic Email-Based Meeting Scheduler

Hugo Liu, Hassan Alam, Rachmat Hartano

*** DRAFT, DO NOT DISTRIBUTE ***

BCL Computers, Inc.
990 Linden Drive, Suite 203
Santa Clara, CA 95050
hugoliu@mit.edu, {hassana, rachmat}@bcl-computers.com

Abstract
BCL Meeting Runner is a software agent that acts as a user's personal secretary by observing the user's incoming emails, identifying requests for meetings, and interacting with the person making the request to schedule the meeting into the user's calendar, on behalf of the user. We describe robust information extraction techniques that are well-suited to the email meeting scheduling domain. We then present the architecture of the robust natural language core of our system, and give a performance evaluation of our first generation system. Our approach to robust meeting request extraction from emails combines the broad coverage of collocation with the robustness of semantic grammars. Using a relatively small collocation-based semantic grammar, we are able to demonstrate an f-measure of .85 for the email meeting request domain.

Introduction
When we think of computers of the future, what comes to mind for many are personal software agents that help us to manage our daily lives, taking on responsibilities such as booking dinner reservations, and ordering groceries to restock the refrigerator. One of the more useful tasks that a personal software agent might do for us is to help manage our schedules – booking an appointment requested by a client, or arranging a movie date with a friend. Because today we rely on email to accomplish much of our social and work-related communication, and because emails are in some senses less invasive to a busy person than a phone call, people generally prefer to request meetings and get-togethers with co-workers and friends by sending them an email message. The sender might then receive a response confirming, declining, or rescheduling the meeting. This back-and-forth interaction may continue many times over until something is agreed upon. Such a task model is referred to as asynchronous email meeting scheduling.

Previous approaches to software-assisted asynchronous email meeting scheduling either require all involved parties to possess common software, such as with Microsoft Outlook, and Lotus Notes, or require explicit user action, such as with web-based meeting invitation systems like evite.com and meetingwizard.com.

In the first approach taken by Microsoft Outlook and Lotus Notes, users can directly add meeting items to the calendars of other users, and the software can automatically identify times when all parties are available. This is very effective, and can be very useful within companies where all workers have common software; however, such an approach is inadequate as a universal solution to email meeting scheduling because a user of the software cannot use the system to automatically schedule meetings with non-users of the software, and vice versa.

The second approach exemplified by evite.com and meetingwizard.com moves the meeting scheduling task to a centralized web server, and all involved parties communicate with that server to schedule meetings. Because all that is required is a web browser, the second approach circumvents the software-dependency limitations of the former. However, a drawback is that this system for meeting scheduling is not automated; it requires users to read the email with the invite, open the URL to the meeting item, and check some boxes. If the meeting details were to change, the whole process would have to repeat itself. It is evident that this approach is not amenable to automation.

Why natural language?
The approach we have taken is to build a personal software agent that can automatically interact with co-workers, clients, and friends to schedule meetings through emails by having the interaction take place in plain and common everyday language, or natural language as it is generally called. Natural language is arguably the most common format a software program can communicate in, because humans are already proficient in this. By specifying natural language as the format of emails that can be understood
and generated by our software agent, we can overcome the problem of required common software (a person who has installed our agent can automatically receive meeting requests and schedule meetings with someone who does not have our agent installed), and the problem of required user action (our agent can interact with the person who requested the meeting by further emails, never requiring the intervention of the user).

This paper is organized as follows. In the first section we discuss our approach to the robust extraction of meeting requests from emails by combining existing information extraction techniques like collocation and semantic grammars. In the second section, we present the architecture of the natural language processing and dialog management components of our software agent. We conclude with an evaluation of our first-generation system, and a discussion of the limitations of information extraction techniques in the automated email meeting scheduling domain, and future work planned for the system.

Robust Techniques for Meeting Request Extraction

Unlike the relative “clean” text found in the Wall Street Journal corpus, text found in emails can be notoriously “dirty”. Email texts often lack proper punctuation, capitalization, tend to have sentence fragments, omit words with little semantic content, use abbreviations and shorthand, and sometimes contain mildly ill-formed grammar. Therefore, many of the deep parsers that can parse clean text well would have a tough time with a dirty text, and are generally not robust enough for this type of input. Thankfully, we do not need such a deep level of understanding for meeting request extraction. In fact, this is purely an information extraction task. As with most information extraction problems, the desired knowledge, which in our case is the meeting request details, can be described by a semantic frame with the slots similar to the following:

- Meeting Request Type: (new meeting request, cancellation, rescheduling, confirmation, irrelevant)
- Date/Time interval proposed: (i.e.: next week, next month)
- Location/Duration/Attendees
- Activity/Occasion: (i.e.: birthday party, conference call)

The task of identifying and extracting meeting request details from emails can be decomposed into classifying the request type of the email as shown in the frame above, and filling in the remaining slots in the frame. In our system, the second task can be solved using the solution to the first problem. We approach the classification of email into request type classes in the following manner: Each request type class is treated as a language described by a grammar. Membership in a language determines the classification. Membership in multiple languages requires disambiguation by a decision tree. If an email is not a member of any of the languages, then it is deemed an irrelevant email not containing a meeting request. We will now describe the properties of the grammar.

A collocation-based semantic grammar

Semantic grammars were originally developed for the domains of question answering and intelligent tutoring (Brown and Burton, 1975). A property of these grammars is that the constituents of the grammar correspond to concepts specific to the domain being discussed. An example of a semantic grammar rule is as follows:

MeetingRequest → Can we get together

DateType for ActivityType

In the above example, DateType and ActivityType can be satisfied by any word or phrase that falls under that semantic category. Semantic grammars are a practical approach to parsing emails for request type because they allow information to be extracted in stages. That is, semantic recognizers first label words and phrases with the semantic types they belong to, then semantic rules are applied to sentences to test for membership in the language. Semantic grammars also have advantage of being very intuitive, and so extending the grammar is simple to understand. Examples of successful applications of semantic grammars in information extraction can be found in entrants to the U.S. government sponsored MUC conferences, including FASTUS system (Hobbs et al., 1997), CIRCUS (Lehnert et al., 1991), and SCISOR (Jacobs and Rau, 1990).

The type of semantic grammar shown in the above example is still somewhat narrow in coverage because the productions generated by such rules are too specific to certain syntactic realizations. For example, the previous example can generate the first production listed below, but not the next two, which are slight variations.

- Can we get together tomorrow for a movie
- *Can we get together tomorrow to catch a movie
- *Can we get together sometime tomorrow and check out a movie

We could arguably create additional rules to handle the second and third productions, but that comes at the expense of a much larger grammar in which all syntactic realizations must be mapped. We need a way to keep the grammar small, the coverage of each rule broad, and at the same time, the grammar we choose must be robust to all the aforementioned problems that plague email texts like omission of words, and sentence fragments. To meet all of these goals, we introduce the idea of collocation to our semantic grammars. Collocation is generally defined as the proximity of two words within some fixed “window” size. This technique has been used in variety of natural language tasks including word-sense disambiguation (Yarowsky, 1993), and information extraction (Lin, 1998). Applying the idea of collocations to our semantic grammar,
we eliminate all except the three or four most salient features from each of our rules, which generally happen to be the following atom types: subjectType, verbType, and objectType. For example, we can rewrite our example rule as the following: (for clarification, we also show the expansions of some semantic types)

MeetingRequest → ProposalType
SecondPersonType GatherVerbType DateType ActivityType

ProposalType → can | could | may | might

SecondPersonType → we | us

GatherVerbType → get together | meet | …

In our new rules, it is implied that the right-hand side contains a collocation of atoms. That is to say, between each of the atoms in our new rule, there can be any string of text. An additional constraint of collocations is that in applying the rules, we constrain the rule to match the text only within a specified window of words, for example, ten words. Our rewritten rule has improved coverage, now generating all the productions mentioned earlier, plus many more. In addition, the rule becomes more robust to ill-formed grammar, omitted words, etc. Another observation that can be made is that our grammar size is significantly reduced because each rule is capable of more productions.

There are however, limitations associated with collocation-based semantic grammars – namely, the words not specified, which fall between the atoms in our rules, can have a unforeseen impact on the meaning of the text surrounding it. Two major concerns relating to meeting requests are the appearance of “not” to modify the main verb, and the presence of a sentence break in the middle of a matching rule. For example, our rule incorrectly accepts the following productions:

• “Could we please not get together tomorrow for that movie? (presence of the word "not")
• “How can we meet tomorrow? I have to go to a movie with Fred. (presence of an inappropriate sentence break)

To overcome these occurrences of false positives, we introduce the notion of negative collocates into our grammar, which we will denote as atoms surrounded by # signs. A negative collocate between two regular collocates means that the negative collocate may not fall between the two regular collocates. We also introduce an empty collocate into the grammar, represented by 0. An empty collocate between two regular collocates means that nothing except a space can fall between the two regular collocates. We can now modify our rule to restrict the false positives it produces as follows:

MeetingRequest → ProposalType 0
SecondPersonType #SentenceBreak# #not#
GatherVerbType #SentenceBreak# DateType #SentenceBreak# ActivityType

Using this latest rule, most plausible false positives are restricted. Though it is still possible for such a rule to generate false positives, pragmatics make further false positives unlikely.

Implications of pragmatics

Pragmatics is the study of how language is used in practice. A basic underlying principle of how language is used is that it is relevant, and economical. According to Grice’s maxim (Grice, 1975), language is used cooperatively and with relevance to communicate ideas. The work of Sperber and Wilson (1986) adds that language is used economically, without intention to confuse the listener/reader.

The implications of pragmatics on our grammar is that although there exists words that could be added between collocates to create false positives, the user will not do so if it makes the language more expensive or less relevant. This important implication is largely validated by the fallout metric in performance evaluations which we will present later in this paper. Our discussion now turns to the implementation of our first-generation system.

System Architecture

Figure 1. In this flowchart, nodes represent processing steps.

The natural language and dialog management components of our software agent passively observe incoming emails, and if a meeting request is detected, the type of request, and the meeting request frame, which includes all the details of the meeting, are automatically passed on to other...
components which interact with the user’s calendar and sends emails to the person requesting the meeting.

Our system architecture has the structure shown in Figure 1.

**Dialog Management**
The dialog management step examines the incoming email header to determine if that email belongs to a thread of emails in which a meeting is being scheduled. To accomplish this, each email determined to contain a meeting request is added to a repository of “dialog context frames” (DCF). A DCF serves to link the unique ID of the email, which appears in the email header, to the meeting request frame that contains the details of the meeting. The slots of the DCF are nearly identical to the meeting request frame, except that it also contains information about the email thread it belongs to. DCFs are passed on to later processing steps, and provide the historical context of the current email, which can be useful in disambiguation tasks. In addition, the most recent request type, as specified in a DCF, helps to determine the allowable next request type states. In other words, a finite state automaton dictates the allowable transitions between consecutive email requests. Figure 2 shows a simplified version of the finite state automaton that does this.

**Normalization**
Because of the dirty nature of email text, it is necessary to clean, or normalize, as much of the text as possible by fixing spelling and abbreviations, and regulating spacing and punctuation.

![Diagram of finite state automaton](image)

**Figure 2.** The nodes in this simple finite state automaton represent request types, and the directed edges represent allowable transitions between the states

So in the above example, an email whose DCF has the request type “meeting declined” cannot be classified as “meeting confirmed” at the present step.

If the repository does not contain a DCF for the current email, a new one is created. The DCF is passed on to the next phase of processing.

**Other normalization tasks are also performed to prepare the text for the next step of processing.**

**Recognition Agents**
Each recognition agent extracts a specific type of information from the email text that becomes relevant to the task of filling the meeting request frames. The resulting semantic types are also used by the parser to determine the meeting request type. For many of the semantic types used by our system, matching against a dictionary of concepts which belong to a semantic type is sufficient to constitute a recognition agent. But other semantic types such as temporal expressions require more elaborate recognition agents, which may use a generative grammar to describe instances of a semantic type.

**Parsing**
Our collocation-based semantic grammar contains no more than 100 rules for each of the request types. The rules are applied to the text with the following constraint: the distance from the first token in the fired rule to the last token in the fired rule must fall within a fixed window size, which is usually ten words. When the language of more than one request type accepts the email text, disambiguation techniques are applied, such as the aforementioned finite state automaton of allowable request type transitions (Figure 2).

**Semantic Interpretation**
After identifying the request type of the email, each request type has a semantic frame associated with it which must be filled as much as possible from the text. Because each slot in the semantic frame has a semantic type associated with it, slot fillers are guaranteed to be atoms. Filling the frame is only a matter of finding the right tokens. In cases where multiple qualifying tokens can fill a slot, distance to the parts of the email responsible which were accepted by the grammar is inversely proportional to the relevance of that token. Therefore, we can disambiguate the attachment of frame slots tokens by word distance.

When the semantic frame is filled, it is sent to other components of the system that apply the necessary actions associated with each request type.

**First generation system**
Our first generation system implements this architecture in Python, taking advantage of the language’s string manipulation and regular expression facilities, and its suitability for rapid application development. The front end of our system is implemented in C++ and Visual Basic using the Microsoft Outlook object model, and the system is initially deployed for Microsoft Outlook 2000 and 2002. This first generation system was implemented over the course of half a year, by two developers. Previous to this
system, a prototype system taking two man months was
demonstrated. The system has been tested and evaluated
internally, and is now in its beta testing phase, with general
deployment as a commercial product in the near future.

**Evaluation**

The performance of the natural language component of our
system was evaluated against a real-world corpus of 1178
emails compiled from the email collections of several people
who reported that they commonly schedule meetings using
emails. The standard evaluation metrics of recall, precision,
fallout, and f-measure were used.

An email is marked as a true positive if it meets ALL of
the following conditions:

- the email contained a meeting request
- the request type was correctly identified by our system
- the request frame was filled out correctly.

Likewise, false positives meet EITHER of the following
conditions:

- the email does not contain a meeting request but was
classified as containing a meeting request
- the request frame was filled out incorrectly

Of the 1178 emails in our test corpus, 176 contain email
requests. Our system discovered 137 true positives and 9
false positives, missing 39 meeting requests. Lacking
evidence from user studies as to which is more important to
the user, precision or recall, we chose the parameter $\beta$
to be one, where precision and recall are given equal weight.
Table 1 summarizes the findings of the evaluation.

We see the evaluation results as positive and
encouraging. The most encouraging result is that fallout is
minimal at 0.8%. The fact that our collocation-based
semantic grammar did not create more false positives
provides some validation to the collocation approach, and
to the implications of pragmatics on the safety of the
collocation.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recall</td>
<td>77.8%</td>
</tr>
<tr>
<td>Precision</td>
<td>93.8%</td>
</tr>
<tr>
<td>Fallout</td>
<td>0.8%</td>
</tr>
<tr>
<td>F-measure ($\beta = 1$)</td>
<td>.851</td>
</tr>
</tbody>
</table>

Table 1: evaluation summary

The recall statistics can be improved by broadening the
coverage of our grammar. As of now, each request type
has fewer than 100 grammar rules, and currently there are
only seven semantic types. We believe that investing more
resources to expand the grammar can boost our recall by
10%. Growing the grammar is fairly easy to do because
rules are simple to add, and each rule can generate a large
number of productions. We plan to do more extensive
evaluations once we have compiled a corpus from our beta
testing.

**Conclusion**

We have built a personal software agent that can
automatically read the user’s emails, detect meeting
requests, and interact with the person making the request
to schedule the meeting. Unlike previous approaches taken
to asynchronous email meeting scheduling, our system
receives meeting requests and generates meeting dialog
e-mail all in natural language, which is arguably the most
portable representation. In this paper we presented a
robust technique for the extraction of meeting requests
from emails by combining semantic grammars with
collocation. We also introduced two unique collocation
operators, the negative collocate and empty collocate
which help to control the tendency of the collocation
grammar to generate certain kinds of false positives. We
argued that such a grammar was most robust for the email
domain, whose text tends to be dirty, and language slightly
ill-formed. Our proposed grammar also has the property in
that it tends to be relatively small, since each rule in the
grammar accounts for many syntactic realizations. We
presented the system architecture of the first generation
implementation, and presented an evaluation of our system
against a test corpus of 1178 emails.

**Limitations**

There are several limitations associated with the approach
taken. One issue is scalability. Unlike systems that use
machine learning techniques to automatically learn rules
from a training corpus, our grammar must be manually
extended. Luckily, the email meeting request domain is
fairly small and contained, and the ease of inputting new
rules makes this limitation bearable.

Another issue is portability. Our grammar and
recognition agents were developed specifically for the email
meeting request domain, so it is highly unlikely that they
will be reusable or portable to other problem domains.

**Future Work**

In the near future, we plan to extend the coverage of the
grammar to include an expanded notion of what a “meeting”
can be. For example many errands such as “pick up the
laundry” could constitute meeting requests.

We would also like to increase the power of the
recognition agents by supplying them with more world
semantic resources, or information about the world. For
example, our current system will understand, “Do you want...
to see a movie tonight?” but it will not be able to understand, “Do you want to see Lord of the Rings tonight?” We can envision providing our recognition agents with abundant semantic resources such as movie names, all of which can be mined from databases on the Web. We have already begun to supply our recognition agents with world knowledge mined out of the Open Mind Commonsense knowledge base (Singh, 2002), an open-source database of approximately 400,000 commonsense facts. By growing the dictionary of everyday concepts our system understands, we can hope to improve the recall of our system.

Acknowledgements

We thank our colleagues at BCL Computers for their feedback on the design of the system, and for their help in evaluating the system. This project was inspired by the Spoken Language User Interface Toolkit project, also at BCL Computers. That project is supported by a U.S. Department of Commerce Advanced Technology Program grant (contract # 70NANB9H3025)

References


evite.com Web Site. Available at http://www.evite.com


Lotus Notes Web Site. Available at http://www.lotus.com/home.nsf/welcome/notes

MeetingWizard.com Web Site. Available at http://www.meetingwizard.com

Microsoft Outlook Web Site. Available at http://www.microsoft.com/outlook/


