Honeydew: Predicting meeting date using machine learning algorithm and adding a meeting to user's calendar from email clients

Chaiyong Ragkhitwetsagul
Carnegie Mellon University
cragkhit@cs.cmu.edu

ABSTRACT

Today scheduling a meeting via email becomes widely used. However, creating a meeting in a user's calendar can be tedious. Honeydew is an intelligent agent, which uses machine-learning algorithms to extract all required information from a meeting email. Then it creates suggestions for the user. The user can verify the suggestions, and proceed to place the meeting in his calendar. The agent can dramatically reduce user's workload of extracting all information from emails and put them in his calendar application. The agent also automatically improves its performances by learning from suggestions that is corrected by human users. This paper discusses an algorithm used for predicting meetings date from content in emails. It also includes the evaluation results of the system. Finally, the paper shows the implementation of the whole system.

General Terms

Algorithms, Performance, Design.

Keywords

Honeydew, machine learning, email, extensions, agent.

1. INTRODUCTION

1.1 Overview

To add a meeting to a calendar application requires a user to fill in many required fields (title, date, time, duration, location), which is tiresome. This is especially true for users who have a lot of meetings and need to store all meetings in a calendar application. Consider a user receives an email asking for attending a meeting at a specific time next week. If he decides to accept the invitation, he can proceed by create a new entry in his Google calendar. He is required to manually insert the meeting title, select meeting date and time. Besides that, he may want to include the location of the meeting into the meeting entry also. All of these steps need the user to go back and forth from his email client and his calendar application. He has to copy all required information and paste them in field by field. Honeydew agent can automate all these steps. From figure 1, when the user receives a meeting email (1) is required only to notify the agent to process the email (2). The agent will generate a suggestion form for the user (3). The user verifies the agent's suggestions (4), and submits the meeting to his calendar (5).

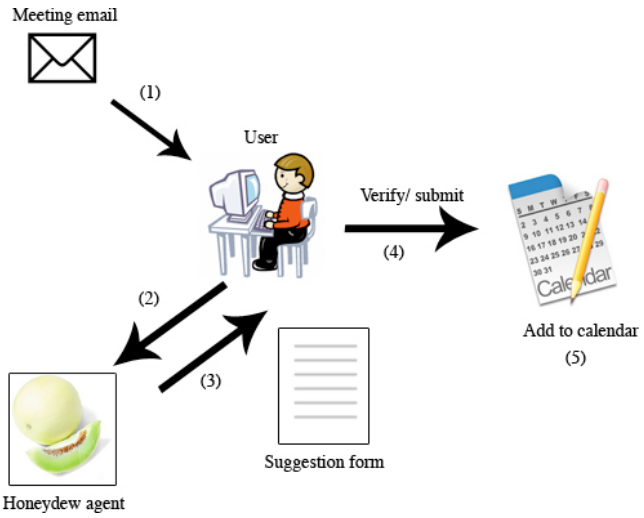

Figure 1. System overview: Honeydew agent receives a meeting email from a user, and generates suggestions form. The user verifies the suggestions and then submits it to his calendar application

Mostly Honeydew agent is mainly based on the framework of VIO agent [20, 23]. Thus, this paper focuses only on a module of Honeydew agent for generating meeting date suggestions. This module consists of Date Property Generator ("DPG") and Date Property Classifier ("DPC). The agent tries to predict the most possible date for the meeting from a date expression in email's content. The algorithm for achieving this is described in the next section. Moreover, we also include the idea of email clients' extensions to complete the process of adding a meeting to calendar in the section 2. The result section provides performance evaluation of the system. In section 5, we discuss problems found and promising future work to be added to the project. The contribution section combines all the work that the writer has contributed to the project. Finally, Appendix A includes the detailed implementation of three email clients' extensions (Mozilla Thunderbird, Gmail on Mozilla Firefox, and Microsoft Outlook 2007), and Honeydew web form, which incorporates Google Calendar API [1].

We use the term "email message" to represent an electronic mail in a user's email server. Honeydew agent means the Java application running on Tomcat server. We use the term Honeydew extensions and extensions interchangeably and they refer to all Honeydew extensions of the three email clients. We call all of them as extension although it might be called differently in its own environment. For example, Mozilla Thunderbird and Firefox call it as extension/add-on, while Microsoft Outlook calls it as add-in. Honeydew web form (suggestion form) represents a web page, which is a response from Honeydew agent. This web page contains the original email message content; a meeting form filled with suggestions from Honeydew agent; and buttons to add the meeting to calendars.

2. METHODS

2.1. Date Property Generator (DGP) and Date Property Classifier (DPC)

General idea is to train the learner (Honeydew agent) to suggest an exact meeting date to its user by using a date expression from a meeting email. That means creating a classifier, which inputs a date expression and outputs a predicted meeting date. This can be done by converting a date into corresponding date properties. Date properties are values that are used to represent a specific date. We have 6 date properties, which include DAY_OF_WEEK, DAY_OF_MONTH, MONTH, DAY_RELATIVE, WEEK_RELATIVE, and YEAR. Then we train many sub-classifiers separately on each date property. DPG and DPC are implemented in Java programming language, so we will use all Java conventions for explaining the design of DPG and DPC from now on.

Date Property Generator is used to generate date properties from a given a message sent date, and an a meeting date. We need both message sent date and actual meeting date because we need to find relation between them for the 3 relative values of date properties: DAY_RELATIVE, WEEK_RELATIVE, and YEAR.

{message sent date, meeting date} → DPG → {date properties}

The algorithms of DPG are explained in more detail in section 3.

Date Property Classifier is a set of classifiers which is used to classify a given date expression, and outputs a probability map for each date property.

{date expression } → DPC → probability map

2.2 Training phase

We perform the training of DPC on 172 email messages. DPC is a container classifier, which contains 6 sub-classifiers. Each of those sub-classifiers is for a particular date property. All the training email messages contain date expression specifying dates of meeting. We label all of the email messages manually. The labeled data is a date expression, message sent date, and its actual meeting date {date expression, message sent date, meeting date}. Each classifier calls DPG to generate date properties out of the actual meeting date, create its own new-labeled data, and train on that new-labeled data.

For example, we have a labeled data from an email message, which looks like this:

{next Wednesday, December 12, 2008, December 17, 2008}

A classifier of DAY_OF_WEEK calls DPG to generate DAY_OF_WEEK property, and then it trains on that value. Thus, the training data for the DAY_OF_WEEK classifier will be:

{next Wednesday, Wednesday}

This is to train the classifier that when it sees "next Wednesday", it is more likely that the DAY_OF_WEEK has a value "Wednesday."

In the same way, a classifier of MONTH calls DPG to generate MONTH property and its training data will be:

{next Wednesday, December}

2.3 Testing phase

The testing phase is opposite to the training phase. Because we do not know the actual meeting date, DPG needs to generate 30 days starting from the email message sent date. These 30 days are call "possible meeting dates." For instance, we have a meeting email, which is sent on December 12, 2008. DPG generates the following possible meeting dates:

December 12, 2008
December 13, 2008
December 14, 2008
…
January 10, 2009

} 30 days

And then each of them contains a set of date properties.

December 12, 2008 – {date properties}

December 13, 2008– {date properties}

December 14, 2008– {date properties}

…

January 10, 2009– {date properties}

Finally, DPC, given a date expression, goes through all possible meeting dates, classifies them based on their date properties, and outputs the probability map for all date properties. Finally, The system picks the best date for the meeting based on the generated probabilities. Figure 2 shows the diagram of the algorithm.

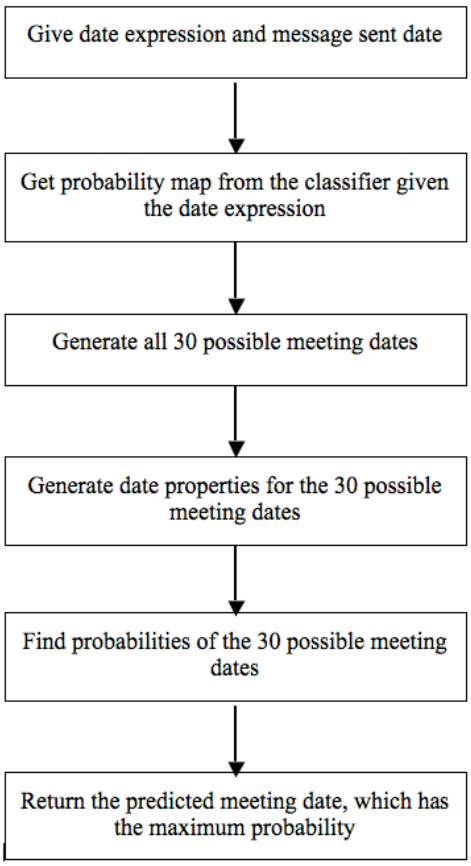

Figure 2 Diagram showing the algorithm of Date Property Generator

Figure 3 illustrates the coordination between DPG and DPC for each one generated possible meeting date.

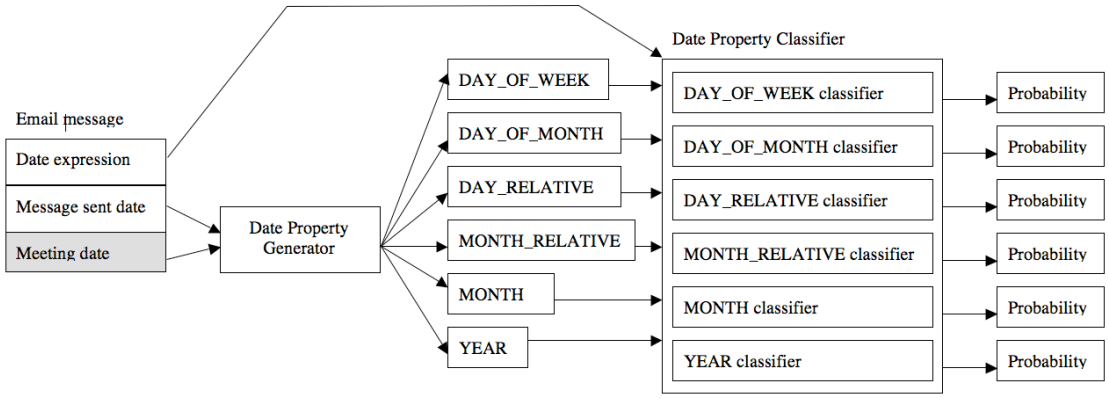

Figure 3. For each email message, DPG generates 6 date properties out of it. DPC uses that date properties in conjunction with date expression from the email message, and produces a probability map. This diagram shows only prediction of one possible meeting date. The real system repeats this step 30 times for each testing email message.

3. ALGORITHMS

3.1 Generate date properties out of a given date, and a meeting date

We define the following values as the date properties for a given date:

DATE_PROPERTIES = { DAY_OF_WEEK, DAY_OF_MONTH, MONTH, DAY_RELATIVE, WEEK_RELATIVE, YEAR}

Each property has its own set of values:

- DAY_OF_WEEK_VALUES = { Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday }
- DAY_OF_MONTH_VALUES = { 1, 2, 3, 4, …, n | n = last day of a given month}
- MONTH = { January, February, March, April, May, June, July, August, September, October, November, December }
- DAY_RELATIVE_VALUES = { TODAY, TOMORROW, DAY_AFTER_TOMORROW, MORE_THAN_NEXT_TWO_DAYS}
- WEEK_RELATIVE_VALUES = { THIS_WEEK, NEXT_WEEK, NEXT_TWO_WEEKS, NEXT_THREE_WEEKS, MORE_THAN_THREE_WEEKS }
- YEAR_VALUES { THIS_YEAR, NEXT_YEAR }

Given two dates (a message sent date and a meeting date), we can generate all the properties from them by based on the message sent date as the starting point. For more understanding, see the following examples:

> Example Given a message sent on 09/28/2008 and we want to find the date properties for the possible meeting date on 09/30/2008.
>
> The result should looks like this:
>
> DateProperties(09/30/2008 | 09/28/2008) = { Tuesday, 30, September, DAY_AFTER_TOMORROW, THIS_WEEK, THIS_YEAR }
>
> Example Given a message sent on 12/30/2008 and we want to find the date properties for the possible meeting date on 1/19/2009, the result should looks like this:
>
> DateProperties(1/19/2009 | 12/30/2008) = { Monday, 19, January, MORE_THAN_NEXT_TWO_DAYS, NEXT_THREE_WEEKS, NEXT_YEAR}

3.1.1 Implementation details

For ease of use, we convert the string representation of a date into a Calendar object. Let say, we use a `Calendar` object *c* to hold the meeting date. We use c_1 for the message sent date and c_2 for the meeting date.

DAY_OF_WEEK – extract directly from c_2.

DAY_OF_MONTH – extract directly from c_2.

MONTH – extract directly from c_2.

DAY_RELATIVE – find difference in number of days of meeting date and message sent date. Then convert the day difference into the DAY_RELATIVE value. We convert two dates into time in milliseconds. This can be done by using `Calendar.getTimeInMillis()` method which outputs the current time as UTC milliseconds from the epoch. Then we find difference in milliseconds and convert it to number of days.

WEEK_RELATIVE – we use the DAY_OF_WEEK value, and the day difference value from the DAY_RELATIVE calculation to find the week relative value.

First, let's define $d_{1,2}$ as the day difference from two dates; D_1 is the number of days until the end of the week of c_1.

WEEK_RELATIVE is equal to:

- THIS_WEEK if $d_{1,2} <= D_1$
- NEXT_WEEK if $D_1 < d_{1,2} <= D_1+7$
- NEXT_TWO_WEEKS if $D_1+7 < d_{1,2} <= D_1+14$
- NEXT_THREE_WEEKS if $D_1+14 < d_{1,2} <= D_1+21$
- MORE_THAN_THREE_WEEKS if $D_1+21 < d_{1,2}$

YEAR – we find year difference *y* between two dates by using the values from c_1 and c_2.

- THIS_YEAR if $y = 0$
- NEXT_YEAR if $y = 1$

3.1.2. Generate all possible meeting dates from a message sent date

We assume that the meeting is going to happen in the next 30 days only. Which is really practical in real world that an email confirming about a meeting will be sent within a month before the exact meeting date. The algorithm looks like this:

1. Use a Calendar object c_0 representing the message sent date as the starting date.
2. Increase the day of c_0 by one *(i.e. $c_0 = c_0 + 1$)*.

3. Create a new Calendar object c_1 and set its time to be a new updated time of c_0.
4. Put c_1 in the result lists.
5. Repeat step 2-4 until we complete all 30 days (and put c_2-c_{30} into the result list).

3.1.3. Find the most possible meeting date by using probabilities from the classifier

For this algorithm, we need two values: date expression, and message sent date. We get the probability map for each date property from the classifier by giving it the date expression. We use the message sent date for generating 30 possible meeting dates. Then we go through each date and calculate its probability and find the maximum one. The algorithm looks like this:

1. We have a date expression D_{exp} and a message sent date (in Calendar format) c_0.
2. Get the probability map M from the classifier by giving D_{exp}.
3. Generate all possible dates c_1-c_{30} from c_0.
4. Generate date properties for those possible dates c_1-c_{30}. We use P_i as a set of date properties of date c_i. So, we get $P_1, P_2, ..., P_{30}$. Each P_i contains 6 properties p_{i1}, p_{i2}, p_{i3}, p_{i4}, p_{i5} and p_{i6}.
5. For each c_i, calculate the product of probabilities of all properties.

$$R_i = M(c_i, p_{i1}) \times M(c_i, p_{i2}) \times M(c_i, p_{i3}) \times M(c_i, p_{i4}) \times M(c_i, p_{i5}) \times M(c_i, p_{i6})$$

6. We keep only the maximum probability each time we calculate.

$$\begin{aligned} &\textit{if } (R_i > Max) \textit{ then } \{ \\ &\qquad Max = R_i \\ &\qquad PredictedDate = C_i \\ &\} \end{aligned}$$

7. When we complete all 30 possible dates, C_i is the most possible date and will be the result of the algorithm.

3.2 Adding meeting emails to user's calendar

Our main goal of Honeydew project is usability. The Honeydew agent should be really helpful in reducing tedious tasks that the user needs to go through. We propose an idea of creating a button in a user's email client. The user uses this button to notify the agent to process any emails that he categorizes as a meeting email (it contains meeting-related content). The extension then sends the whole email content to Honeydew agent. After the agent processes the email and generates suggestions for it, it sends back the Honeydew web form URL to the extension. The extension redirects the user to Honeydew web form

in web browser, which has been filled with suggestions corresponding to that meeting. The user can check correctness of the suggestions, and correct them if he finds that some of them are wrong. Finally, he proceeds by submitting the form and the meeting will be added to his calendar.

We provide various choices for users in selecting email clients and calendar applications. Our current system supports the following email clients: Mozilla Thunderbird, Gmail on Mozilla Firefox, and Microsoft Outlook 2007. For calendar applications, we support Google Calendar and Microsoft Outlook. Figure 4, 5, 6 show Honeydew button on Mozilla Thunderbird, Gmail, and Outlook respectively.

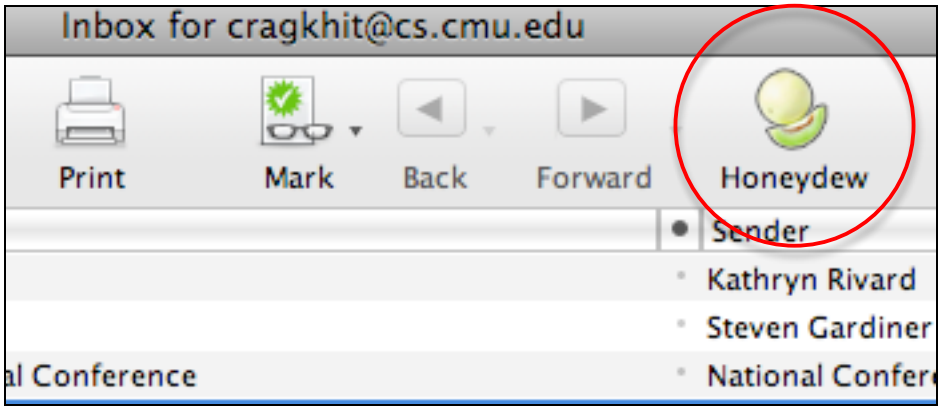

Figure 4. Honeydew button appears in Thunderbird toolbar. A user can request the agent to process a selected email by pressing the button

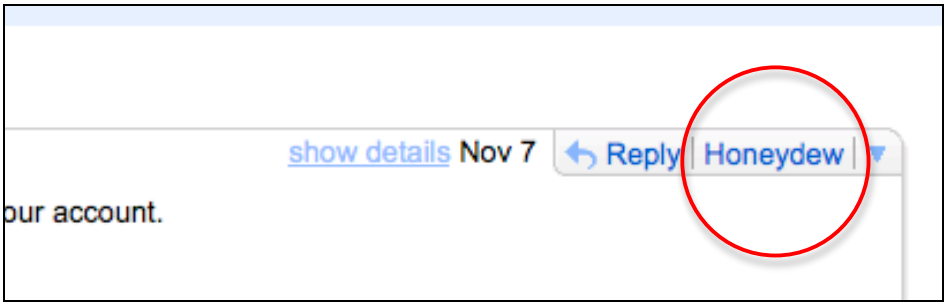

Figure 5. Honeydew button in Gmail

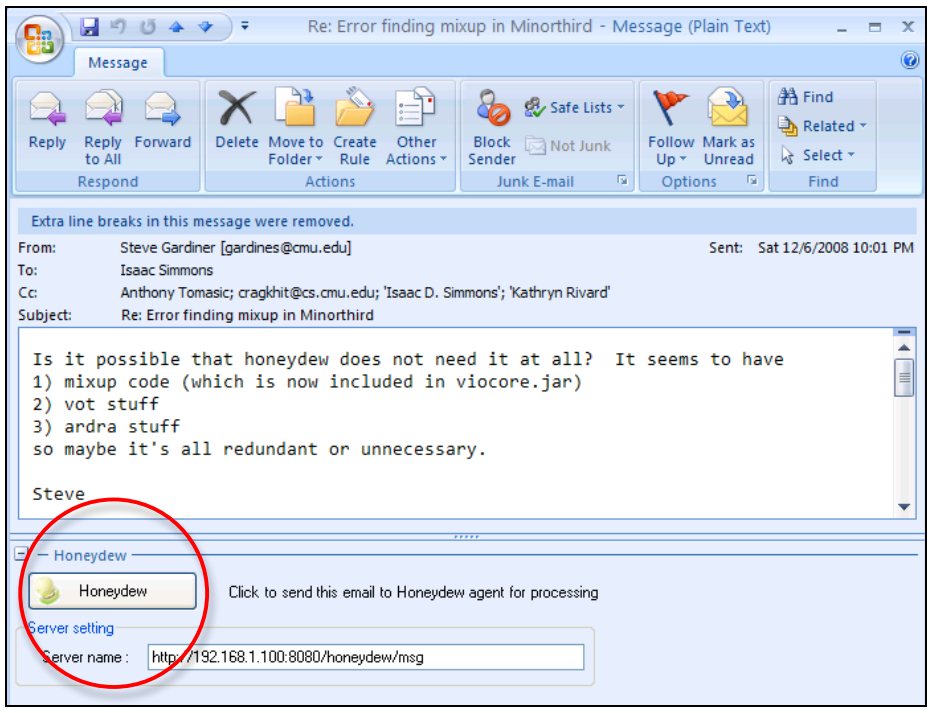

Figure 6 Honeydew button in Outlook 2007

A user can select a meeting email message, and press Honeyew button (on any extension he chooses). After pressing the button, the Honeydew web form will show up with suggestions already filled in (figure 7).

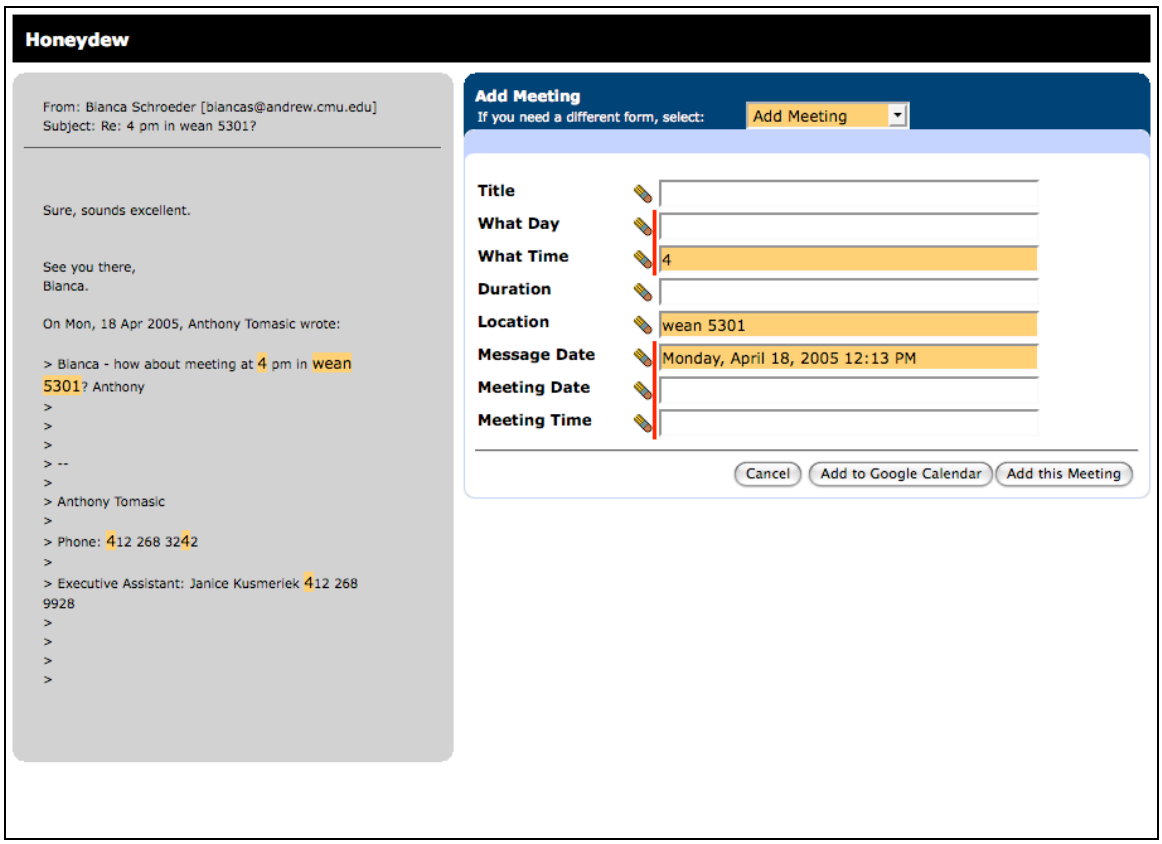

Figure 7. Honeydew web form with suggestions from Honeydew agent

Finally, when the user presses "Add to Google calendar" button. The meeting will be automatically added to the user's calendar (figure 11). However, the user needs to create a calendar named "honeydew" in his account before using this feature.

Note that, for accessing the user's Google Calendar. We need to use the Google API for authenticating the user. The Google Calendar API provides a different cookie from normal Google single-sign-on cookie. That means even the user has already logged into his Google account, he still needs to authenticate himself to the Google Calendar API again. There are two possible scenarios:

1. User has not logged in to his Google account and Google Calendar API.

   He will see the Google login page asking him to login (see figure 8).

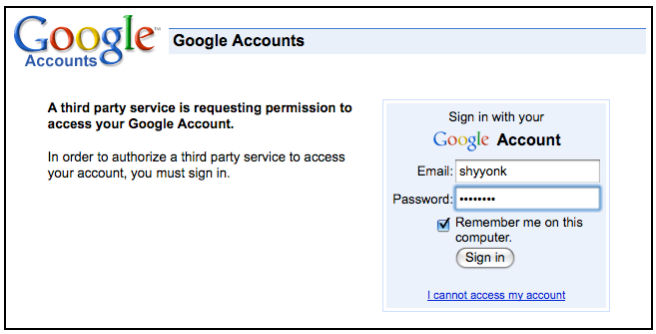

Figure 8 Google account login page

Then, he will be asked for granting access to his calendar application (figure 9).

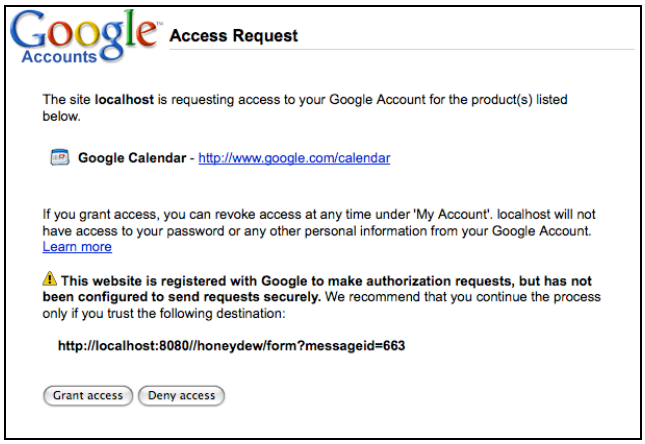

Figure 9 Google authentication page

The authentication process will complete when the user presses "Grant access" button. A cookie (called "token" in Google API page) will be also placed in the user's browser. From now on Honeydew agent can create a new entry in the user's calendar without requiring further authentication.

2. User has already logged in to his Google account but not logged in to Google Calendar API.

   It is possible to logout from Google Calendar account only. We do not provide a way to do this in our Honeydew web from page, yet the user might access other web applications, which include Google Calendar API logout capability. In this case, the user will be redirected to the Google Access Request page (figure 9) only. After the grant access, the authentication process is completed.

When the authentication process completes, Google Calendar API will redirect the user back to the Honeydew web form page. This is a requirement of the API itself, and it is one of our main issues to be fixed. The user needs to fill in the form and press the "Add to Google calendar" button again. We already included this in our future work. The Honeydew web form page will show a notification saying that the meeting has added to the user's calendar (figure 10).

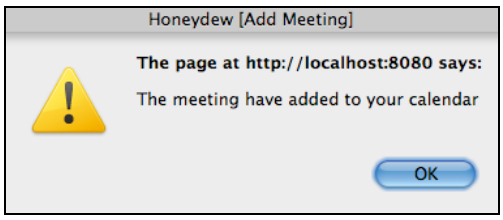

Figure 10 Notification of adding a meeting to Google Calendar from Honeydew agent

The user can now see the meeting appears in his Google Calendar page.

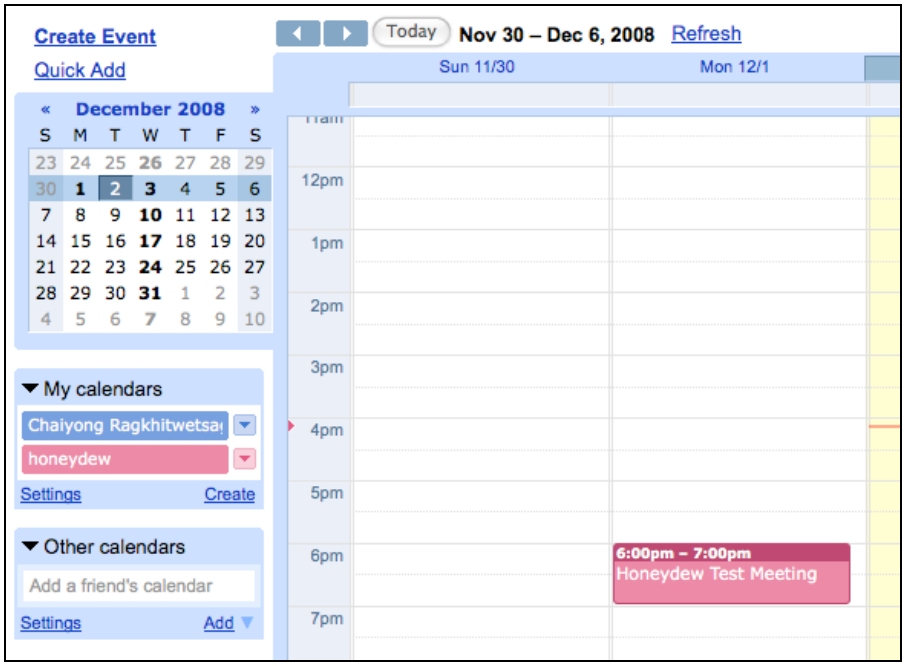

Figure 11. Honeydew adds the meeting to the user's Google calendar

4. RESULTS

4.1. Performance of Honeydew agent on training data without DPG and DPC

For this first step, we trained the Honeydew agent (without Date Property Classifier) on 50 selected email messages, which all contain meeting related contents. We have the following 7 annotators, excluding the Date Property Classifier, which we use for classifying Meeting Date:

- Meeting Title – the title of the meeting. Can be blank if it's not stated in the email message.
- What Day (or Day Phrase) – the date expression specifying the day of the meeting. For example, this Monday, or next Thursday.
- What Time (or Time Phrase) – the time expression specifying the meeting time. For example, 3 – 4 PM, or 8 am.
- Duration – duration of a meeting. Can be blank if not stated.
- Location – meeting location. Can be blank if not stated.
- Message Date – the date that the email is sent. This is auto-filled in by the Honeydew agent using the date from email's header.
- Meeting Time – meeting time in a standard format (3:00 – 4:00 PM).

The results are shown in Table 1.

- means no text filled in the form, + means Correct text filled in the form, Δ means partially correct text filled in the form, and O means incorrect text filled in the form

Table 1. Results of running Honeydew agent against 50 training email messages

Agent's result/ Actual value	Title	What day	What time	Duration	Location	Message Date	Meeting Date	Meeting Time
- / -	15	1	0	26	5	0	0	0
- / +	2	2	7	19	2	0	50	37
+ / -	0	0	0	0	0	0	0	0
+ / +	25	44	42	0	40	50	0	0
Δ / -	0	0	0	0	0	0	0	0
Δ / +	1	2	1	0	3	0	0	13
O / -	2	0	0	0	0	0	0	0
O / +	5	1	0	5	0	0	0	0

4.1.1. Analysis

From the results, our annotators perform nicely on Meeting Title, What Day, What Time, and Location. The agent performs poorly on Meeting Time by generating only 13 results. However, we can improve the agents' performance by training it with more training data. Because we ran the training on only 50 email messages, that is not large enough to obtain accurate annotators. The results of Honeydew agent trained on 172 email messages are included in section 4.3.

4.2. Performance of DPG and Date Property Classifier (DPC) on training data

We trained the Date Property Classifier on 171 email messages. All of them contain meeting related contents, and proposed date of meeting. After training, we evaluated its performance by running it against the training set. The result is shown in Table 2.

Table 2. Result of running Date Property Classifier on training email messages

Type of email messages	Number of email messages
Training messages	171
Testing messages	171
Correct predicted meeting dates	167
Incorrect predicted meeting dates	4

The classifier produces reasonable results when running against its training email messages. However, there are 4 email massages misclassified. We need to investigate on this as a future work.

4.3. 10-fold cross validation on 172 training email messages

We finally perform an evaluation on training email messages using 10-fold cross validation. The email messages are separated into 10 sets equally, and then train the learners on 9 sets and test on another 1 set. We use Apache Ant target to automate some evaluating steps. The steps are shown below:

1. Load Honeydew schema into the database (clear all existing data).
2. Import a Honeydew form log file, which contains 9 sets of data into the database.
3. Train the learners.
4. Export a Honeydew learned log file, which contains binary data of trained learners. This is for further uses.
5. Run the learners against the 1 set of testing data.
6. Repeat step 1-5 until completing all 10 sets.

4.3.1 Results

Round 1

Table 3 Results of round 1

Message	Meeting Title	Day Phrase	Time Phrase	Duration	Location	Message Date	Meeting Date	Meeting Time
Correct	4	6	6	18	7	18	8	3
Incorrect	14	12	12	0	11	0	10	15
Total	18	18	18	18	18	18	18	18
% Correct	22.22	33.33	33.33	100.00	38.89	100.00	44.44	16.67

Round 2

Table 4 Results of round 2

Message	Meeting Title	Day Phrase	Time Phrase	Duration	Location	Message Date	Meeting Date	Meeting Time
Correct	1	6	9	17	9	17	6	2
Incorrect	17	12	9	1	9	1	12	16

Total	18	18	18	18	18	18	18	18
% Correct	5.56	33.33	50.00	94.44	50.00	94.44	33.33	11.11

Round 3

Table 5 Results of round 3

Message	Meeting Title	Day Phrase	Time Phrase	Duration	Location	Message Date	Meeting Date	Meeting Time
Correct	2	4	7	17	10	17	6	2
Incorrect	15	13	10	0	7	0	11	15
Total	17	17	17	17	17	17	17	17
%Correct	11.76	23.53	41.18	100.00	58.82	100.00	35.29	11.76

Round 4

Table 6 Results of round 4

Message	Meeting Title	Day Phrase	Time Phrase	Duration	Location	Message Date	Meeting Date	Meeting Time
Correct	5	6	7	17	8	17	7	0
Incorrect	12	11	10	0	9	0	10	17
Total	17	17	17	17	17	17	17	17
%Correct	29.41	35.29	41.18	100.00	47.06	100.00	41.18	0.00

Round 5

Table 7 Results of round 5

Message	Meeting Title	Day Phrase	Time Phrase	Duration	Location	Message Date	Meeting Date	Meeting Time
Correct	7	6	8	17	9	17	3	0
Incorrect	10	11	9	0	8	0	14	17
Total	17	17	17	17	17	17	17	17
%Correct	41.18	35.29	47.06	100.00	52.94	100.00	17.65	0.00

Round 6

Table 8 Results of round 6

Message	Meeting Title	Day Phrase	Time Phrase	Duration	Location	Message Date	Meeting Date	Meeting Time
Correct	6	10	7	16	12	17	9	0
Incorrect	11	7	10	1	5	0	8	17
Total	17	17	17	17	17	17	17	17
%Correct	35.29	58.82	41.18	94.12	70.59	100.00	52.94	0.00

Round 7

Table 9 Results of round 7

Message	Meeting Title	Day Phrase	Time Phrase	Duration	Location	Message Date	Meeting Date	Meeting Time
Correct	4	3	8	17	9	17	4	0
Incorrect	13	14	9	0	8	0	13	17
Total	17	17	17	17	17	17	17	17
%Correct	23.53	17.65	47.06	100.00	52.94	100.00	23.53	0.00

Round 8

Table 10 Results of round 8

Message	Meeting Title	Day Phrase	Time Phrase	Duration	Location	Message Date	Meeting Date	Meeting Time
Correct	4	8	5	17	9	17	10	0
Incorrect	13	9	12	0	8	0	7	17
Total	17	17	17	17	17	17	17	17
%Correct	23.53	47.06	29.41	100.00	52.94	100.00	58.82	0.00

Round 9

Table 11 Results of round 9

Message	Meeting Title	Day Phrase	Time Phrase	Duration	Location	Message Date	Meeting Date	Meeting Time
Correct	0	4	9	17	7	17	5	1
Incorrect	17	13	8	0	10	0	12	16
Total	17	17	17	17	17	17	17	17
%Correct	0.00	23.53	52.94	100.00	41.18	100.00	29.41	5.88

Round 10

Table 12 Results of round 10

Message	Meeting Title	Day Phrase	Time Phrase	Duration	Location	Message Date	Meeting Date	Meeting Time
Correct	1	5	7	17	8	17	6	1
Incorrect	16	12	10	0	9	0	11	16
Total	17	17	17	17	17	17	17	17
% Correct	5.88	29.41	41.18	100.00	47.06	100.00	35.29	5.88

Average across 10 rounds

Table 13 Average of 10-round results

Message	Meeting Title	Day Phrase	Time Phrase	Duration	Location	Message Date	Meeting Date	Meeting Time
Total Correct	34	58	73	171	88	171	64	9
Total incorrect	138	114	99	1	84	1	108	163
Total Messages	172	172	172	172	172	172	172	172
% Correct	19.77	33.72	42.44	99.42	51.16	99.42	37.21	5.23
% Incorrect	80.23	66.28	57.56	0.58	48.84	0.58	62.79	94.77

4.3.2 Analysis

From the results, the efficiency of the learners still needs to be improved, especially the extractor for Meeting Title (%18 correct), and Day Phrase (%32 correct). The Day Phrase classifier is important because the Meeting Date value depends on the result of Day

Phrase field. Note that Message Date is automatically extracted from the email header, so it is always correct.

Unfortunately, we had many email messages from the same thread and all of them have almost the same content. This affects the evaluation results since some contents in the testing email messages have already seen exactly the same in the training email messages. As a result, most of the suggestions are correct on these testing email messages.

Moreover, we observe that the extractor for What Time tends to pick the last value it finds. It will be more correct if we can modify it to select the first one instead. This is because new contents in replied mail are mostly inserted on the top of the original message contents. We also find some duplication among training email messages. This results in inaccuracy of the learner's performance measurement. We need to remove the duplicated messages in future use. Finally, major improvement could be achieved by increasing the number of training email messages.

5. DISCUSSION

5.1. Problems

5.1.1. Memory leak problem

Honeydew agent was found that it ran out of 1GB heap size after processing 10 email messages. This is not a common amount of memory usage. After an investigation in a memory snapshot file using YourKit Java profiler[1] tool, we found that the Annotator class was the biggest class. Figure 12 shows the memory snapshot captured using Yourkit Java profiler tool:

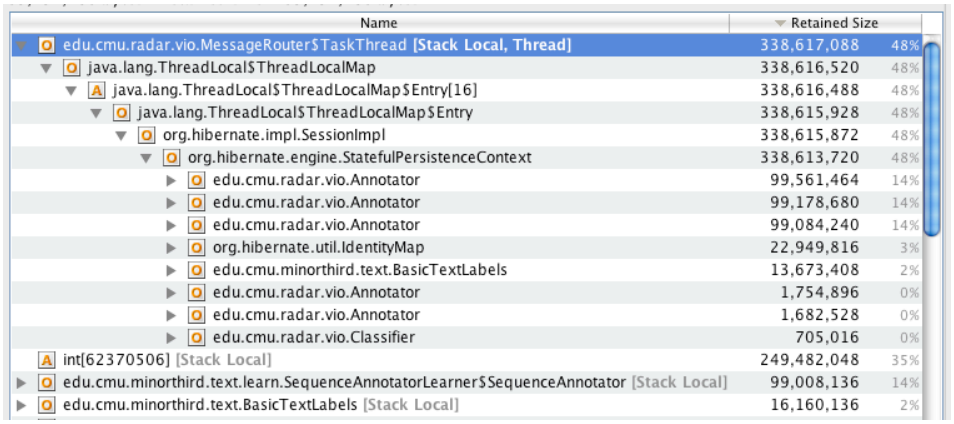

Figure 12. Memory snapshot file shows the biggest objects, which consume most of memory in Honeydew

[1] YourKit Java profiler tool is a commercial tool for testing and solving problem of Java programs. It can create a snapshot file of Java heap for debugging. It also provides user interfaces to look into the cause of the problem.

We can obviously see that the TaskThread, which is an inner class of MessageRouter class, is the biggest object. The reason is because it contains all the annotators in an array list, and 3 annotators have approximately 100 megabytes in size (notice that we also have another 100 megabyte annotator outside the TaskThread). After we performed investigation on Honeydew Java code, we found that the call hierarchy is as follows:

HoneydewAgent >

MessageRouter >

HoneydewWorkflow >

VioExtractor >

VioExtractor.addBodyExtraction() >

VioExtractor.callExtractorAnnotator() >

savedAnnotator.annotate(labels)

We ended up finding that the final class is BeamSearcher inside Minorthird [13] jar file. After a discussion with Frank Lin who is the current Minorthird maintainer, we found that each BeamSearcher caches the instances in memory, and multiple VPCMMLearners means multiple caches. Frank has made caching optional and off by default. We replaced the old minorthird.jar with a newly updated one, and it solved the problem. Previously, Honeydew agent kept increasing its memory usage and almost never released it back to the heap. We can see from figure 13 that, using the new minorthird.jar, the memory has been claimed back to the heap every time that Honeydew agent finishes processing one email message.

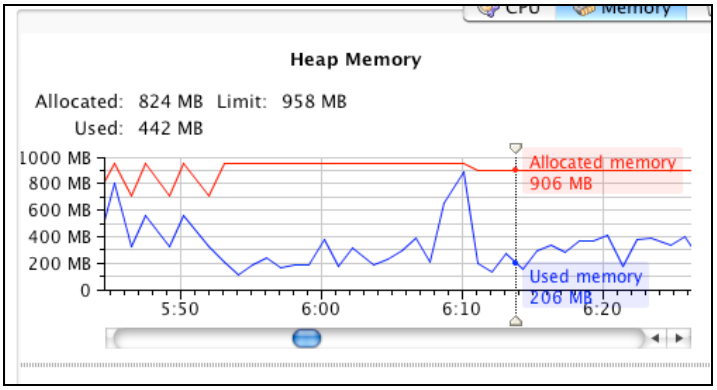

Figure 13 Memory snapshot showing that Honeydew agent releases memory back frequently

5.1.2. Sending email content to Honeydew agent servlet

The first idea of sending email content from email client is by using HTTP POST method while launching the web browser to Honeydew web from URL. However, we found out that it is not possible to launch a web browser and send post data to it. A workaround on this (see figure 14) is by posting the data (email content) to Honeydew agent servlet using

XML HTTP POST request [16] (1). The agent processes the data and sends back the URL for its corresponding form (2). Then, the extension launches the system's default web browser using that URL (3, 4).

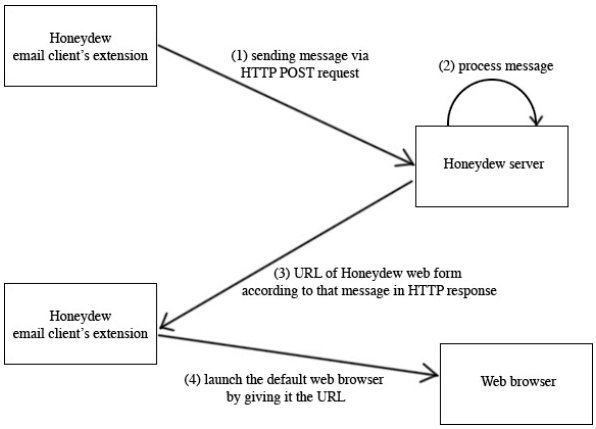

Figure 14. Communication between Honeydew email client extension and Honeydew server

The implementation steps and instructions of how to use all the three extensions (Thunderbird, Gmail, and Outlook) can be found in Appendix A.

5.1.3 Google Greasemonkey API

We decided to implement a Honeydew button on Gmail because there is a majority of users using Gmail nowadays. The easiest way to retrieve and modify the content of Gmail is via GmailGreasemonkey10API [9]. A Google employee is the creator of this API. However, there are some flaws in the API, which makes it not working properly. We use a method called `onViewChangeCallback()` to add the Honeydew button whenever the users change the view to "conversation pane", which means the reading pane that email content appear. This method is supposed to be called every time that the user changes his view to conversation pane, but it is not.

This problem is solved by simply traversing the DOM elements in Gmail page. The code is adopted from one Greasemonkey [2][7] script called Filter Assistant [10]. We modified the code, which places a button near Gmail's Reply button. Then we include our own codes to sending a selected email to Honeydew server, and launch the default web browser after receiving the response back. Further implementation details are in Appendix A.

5.1.4. Modifying the VIO form page

We rely on the VIO form page to use as the Honeydew web form, but still need some flexibility for our own specifications. However, the way VIO generates the web page is through Java and Javascript codes. We decide to put two more buttons in the VIO form

[2] GreaseMonkey is a Mozilla Firefox extension, which provides APIs for developers to write their own Javascript for modifying the browser's behaviors on any specific web page.

page: “Add to Google Calendar” button, and “Add to Outlook” button. What the two buttons do is adding a meeting to user’s calendar.

5.2. Future work

Fixing redirecting problem in Google Calendar API

We need to avoid the redirecting from Google authentication page back to the Honeydew web form page because it needs the user to fill in the form, and press “Add to Google Calendar” twice. We are discussing of using an intermediate page, which performing authentication to Google Calendar. When it is redirected back from Google authentication page with an authenticated token (or cookie), it will automatically add the meeting to the calendar without additional user intervention.

Increasing accuracy of DPG and DPC

DPG and DPC generated 4 misclassified meeting dates. This is probably because the small numbers of training email messages and all of them do not contain date expressions of a whole year. This can importantly affect the accuracy of the DPC since the probabilities of some months are zero (the classifiers never see those months before). This is also true to DAY_RELATIVE, MONTH_RELATIVE, and YEAR_RELATIVE. Training the classifiers with more training email messages should improve the efficiency of the classifiers.

6. CONTRIBUTIONS

These are my contributions to Honeydew project in Fall 2008 semester (August – December).

Email annotation for training the Honeydew agent

I performed annotation on 50 selected email messages. This is an important step to test accuracy of our learners. The annotation process was done by running Honeydew agent without any learners on those email messages, and fill in all the values in Honeydew web form without any suggestions. We use the annotations from this step to train the learners.

Evaluation of Honeydew agent’s performance

I evaluated the Honeydew agent’s performance by running it against the 50 training email messages. The results were collected and presented in Table 1. Moreover, I performed the 10-fold cross validation on 172 email messages. The results are presented in section 4.3.1.

Creating Date Property Generator class

Date Property Generator is a vital part of Date Property Classifier (a classifier for *Meeting Date*). The idea of using date properties to represent a specific date is from Isaac Simmons, one of our project members. This idea is implemented using Java as a class called DatePropertyGenerator. Its algorithm is explained in Method section.

Evaluation of the Date Property Classifier performance

I ran the Date Property Classifier on 171 email messages, which contain date expression. The evaluation results are in Table 2.

Creating Honeydew Thunderbird extension

Most of the works have been done in Javascript, and XUL language. The detailed implementation steps are in Appendix A.

Creating Honeydew Gmail extension

This extension is written in Javascript, and GreaseMoneky API. See Appendix A for more details.

Creating Honeydew Outlook extension

Outlook extension is created as a Microsoft Office add-in using Visual Studio 2008. More information can be found in Appendix A.

Modify the VIO form page

I figured out a way to include a new button into the VIO form page. This can be done using Javascript. With many helps from Steve Gardiner, I was able to include a button to add a meeting to Google calendar in the page.

Solving the memory leak problem

I worked on many ways to trace the cause of the memory leak problem. This includes using YourKit Java profiler, switching between loading learners from Hibernate and Memory, change types of learners, and updating the Minorthird jar file.

7. CONCLUSION

Honeydew is a machine-learning agent to help a user schedule a meeting directly to their calendar from an email message. It can assist the user in doing tedious tasks as extracting all meeting information out of email messages. Honeydew agent can improve its accuracy while it is serving its user by training on the labeled information input by the user. Thus, this is an efficient way of improving the learner's performance without requiring a large number of training data. Date Property Generator and Date Property Classifier are modules in the Honeydew agent, which their task is predicting a meeting date from a date expression appearing in an email message. They perform the classification based on a novel idea of date properties from a given date expression. The evaluation results show that Honeydew agent can perform well on suggesting location and time phrase, but still has problems on meeting title, day phrase, meeting date, and meeting time. With future improvement, Honeydew agent is a promising solution of an intelligent agent, who assists a user to perform its task.

8. ACKNOWLEDMENTS

Thanks to my program director, Anthony Tomasic for contributing as my advisors in this project. Also thanks to Steve Gardiner, Isaac Simmons, and Frank Lin for all of your supports.

9. REFERENCES

[1] API Developer's Guide: The Protocol - Google Calendar APIs and Tools - Google Code. Available from http://code.google.com/intl/th/apis/calendar/docs/2.0/developers_guide_protocol.html; Accessed on 11 November 2008.

[2] Building a Thunderbird extension – MDC. Available from https://developer.mozilla.org/en/Building_a_Thunderbird_extension; Accessed on 9 October 2008.

[3] Creating toolbar buttons – MDC. Available from https://developer.mozilla.org/en/Creating_toolbar_buttons; Accessed on 9 October 2008.

[4] Customizing a Ribbon for Outlook. Available from http://msdn.microsoft.com/en-us/library/bb398246.aspx; Accessed on 5 November 2008.

[5] Customizing the Ribbon in Outlook 2007. Available from http://msdn.microsoft.com/en-us/library/bb226712.aspx; Accessed on 5 November 2008.

[6] C# free example : How to launch an instance of the default browser and load a web page in it? | consultant developer. Available from http://zamov.online.fr/EXHTML/CSharp/CSharp_636597.html; Accessed on 1 December 2008.

[7] Dive Into Greasemonkey. Available from http://diveintogreasemonkey.org/; Accessed on 22 October 2008.

[8] Getting Started Programming Application-Level Add-Ins. Availabel from http://msdn.microsoft.com/en-us/office/ms268878.aspx; Accessed on 4 November 2008.

[9] GmailGreasemonkey10API - gmail-greasemonkey - Google Code - API reference for version 1.0 of the experimental Gmail Greasemonkey API. Available from http://code.google.com/p/gmail-greasemonkey/wiki/GmailGreasemonkey10API; Accessed on 27 October 2008.

[10] Gmail Filter Assistant v0.19 for Greasemonkey. Available from http://userscripts.org/scripts/show/7997; Access on 31 October 2008.

[11] How to get the content of selected message? - mozillaZine Forums. Available from http://forums.mozillazine.org/viewtopic.php?f=19&t=274891&start=0&st=0&sk=t&sd=a; Accessed on 17 October 2008.

[12] nsIWebNavigation. Available from https://developer.mozilla.org/En/NsIWebNavigation; Access on 17 October 2008.

[13] Minorthird Documentation. Available from http://minorthird.sourceforge.net/; Accessed on 10 October 2008.

[14] Office UI Customization. Available from http://msdn.microsoft.com/en-us/office/bf08984t.aspx; Accessed on 5 November 2008.

[15] Outlook Add-ins with Visual Studio Tools for Office. Available from http://www.outlookcode.com/article.aspx?ID=42; Accessed on 4 November 2008.

[16] Post data to window – MDC. Available from https://developer.mozilla.org/En/Code_snippets/Post_data_to_window; Accessed on 17 October 2008.

[17] Programming Application-Level Add-Ins. Available from http://msdn.microsoft.com/en-us/office/bb157876.aspx; Accessed on 5 November 2008.

[18] Thunderbird Help: How To Manage Profiles. Available from http://www.mozilla.org/support/thunderbird/profile; Accessed on 9 October 2008.

[19] Thunderbird – MDC. Available from https://developer.mozilla.org/En/Extensions:Thunderbird; Accessed on 20 October 2008.

[20] Tomasic A., Simmons I., and Zimmerman J. 2007. Learning Information Intent via Observation. Proceedings of the International World Wide Web Conference (WWW). DOI= http://www.cs.cmu.edu/~tomasic/doc/2007/TomasicSimmonsZimmermanWWW2007.pdf

[21] User Script Compiler. Available from http://arantius.com/misc/greasemonkey/script-compiler; Accessed on 10 November 2008.

[22] XPCNativeWrapper – GreaseSpot. Available from http://wiki.greasespot.net/XPCNativeWrapper; Accesed on 26 October 2008.

[23] Zimmerman J., Tomasic A., Simmons I., Hargraves I., Mohnkern K., Cornwell J., and McGuire R. M. 2007. VIO: a mixed-initiative approach to learning and automating procedural update tasks. Proceedings of the Conference on Computer/Human Interaction (CHI). DOI= http://www.cs.cmu.edu/~tomasic/doc/2007/ZimmermanEtAlCHI2007.pdf

Appendix A

Thunderbird extension

The Honeydew Thunderbird extension (will be called only "extension" for later use) is written by Javascript and XML User Interface Language ("XUL"). Mozilla, and other developer forums provide a very good tutorial for developing a Thunderbird extension [2, 3, 18]. This appendix focuses on an overview development of Honeydew extension. More detail of how to create an extension can be found in Mozilla's website [19].

The layout of contents in the extension package is as followings:

```
honeydew.xpi:

                /install.rdf

                /defaults/

                /defaults/preferences/defaults.js

                /chrome.manifest

                /chrome/

                /chrome/content/

                /chrome/content/honeydew.js

                /chrome/content/preferences-window.js

                /chrome/content/honeydew.xul

                /chrome/content/preferences-window.xul

                /chrome/content/general.xul

                /chrome/locale/

                /chrome/locale/en-US/honeydew/honeydew.dtd

                /chrome/locale/en-US/honeydew/preferences-window.dtd

                /chrome/skin/*.css,*.png
```

The extension adds a Honeydew button onto the menu bar. This button is attached to the Thunderbird main user interface by XUL language. Then the button binds with back-end Javascript codes. When a user presses the button, the Javascript code is invoked. The extension uses Thunderbird internal library to receive the content of a current selected message [11]. Then it creates a HTTP Asynchronous POST request to Honeydew server. The Javascript code provides a callback function for the server to send a response back. The server processes the message and sends back a Honeydew web from URL of the processed message. The extension then proceeds and launches the system default web browser giving it the URL [12](as describe in section 5.1.2.).

Main Javascript codes reside in /chrome/content/ folder. The /chrome/locale/ directory contains a resource file of text appearing in the user interface. This allows us to localize

the extension easily by adding new resource files. /chrome/defaults/ stores the default configuration value of the server's name. A user can change this default value from the extension preferences menu found in Tools > Add-ons > Extensions > Honeydew > Preferences or Tools > Honeydew > Preferences. The preferences window will show up as in figure 15.

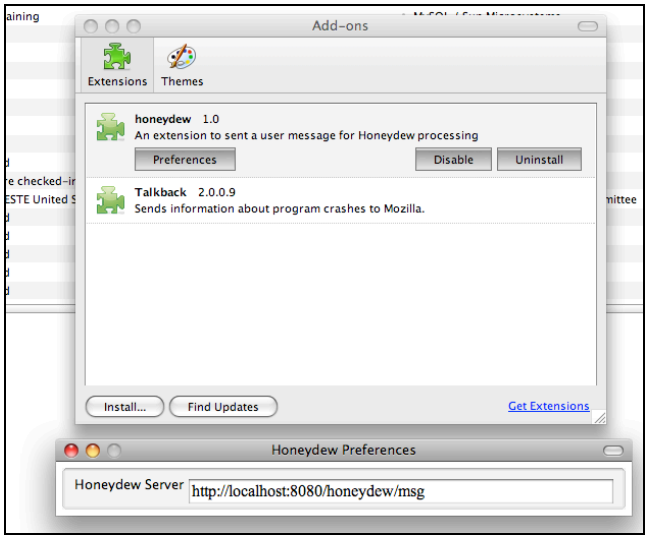

Figure 15. Honeydew preferences window

Gmail extension

Honeydew Gmail extension is slightly different from Thunderbird extension. Although it is an extension, but its purpose is to modify the Gmail's page content not the web browser. We consider Greasemonkey [7] as a versatile tool for achieving our goal. It provides many handy APIs to manipulate page content using Javascript. Greasemonkey requires only a Javascript file, which contains specific domain names that the code will be excecuted. Our Greasemonkey code adds a Honeydew button by manipulating the page's DOM document [22]. The button functions in the same way as Thunderbird's extension. One difference is the content that is sent to the server is in XML format. This is because the email message content displayed in Gmail is HTML formatted. The extension delegates the task of parsing XML to the server.

We found it very difficult to locate an exact location in DOM document in Gmail page. That is because Gmail uses a lot of Javascript and most of all HTML elements are inserted at run time. So, the ordinary command like `document.getElementById()` does not return any element at all. Besides that, the id of each element is also randomly generated every time. This makes it harder to uniquely get one element using its id name. The solution for this is by using a common text appearing on the page. We need to select a text, which is never changed, such as most of the menus. We decide to put the button after the "Reply" button in conversation pane (see figure 5). The Javascript code is adopted from one Greasemonkey code called "Filter Assistant [10]." We use its method to insert the button on that location, and we modify the code that is executed to be our own codes.

Finally, for ease of use, we compile the GreaseMoneky script into a standalone extension using a Greasemonkey compiler [22]. The extension also incorporates the server's preferences as in Thunderbird extension.

Outlook extension

Microsoft Visual Studio 2008 (will be used as VS2008) provides an effortless way to create an add-in (extension) for its Office suite. To create an Outlook extension, we can choose directly by creating a new C# project from VS2008 [4, 5, 6, 8, 14, 15, 17]. We will get the whole required components of the extension. We modify the main file to perform the same way as the two previous extensions. VS2008 also can create an installation file. Users can execute the installation file and the extension will be integrated into Outlook 2007 automatically. They can remove it from Add/Remove Programs menu.

Google calendar API

Google has its own tutorial page for using the Google calendar API [ref]. We choose to use Javascript for communicating with the API. The major reason is because we do not want to keep any user information to make our system most simple. By using Javascript, we connect to the API within the browser. We can use cookie mechanism to help us keeping the user's state. The API returns a token to a new user at the first time he authenticate himself. That token will be kept in the browser for future uses. The steps of connecting to Google calendar API is shown below:

1. Load Google Javascript library to the page by using:

   ```
   google.load("gdata", "1")
   ```

   After this, we can use all the APIs.

2. Check whether the user has already logged in by calling:

   ```
   google.accounts.user.checkLogin(scope)
   ```

   If the user is not logged in, the function returns null. Otherwise, the function returns an existing token.

   We can redirect user to the login page using:

   ```
   google.accounts.user.login(scope)
   ```

   This will be redirected to the login page and performing logging in. This function returns a new token (cookie), which will be kept in the browser.

3. We whether there is a "honeydew" calendar in the user's account by:

   ```
   // Create the calendar service object

   var calendarService = new
   google.gdata.calendar.CalendarService('GoogleInc-jsguide-1.0');

   ...

   ...
   ```

```
// Submit the request using the calendar service object

calendarService.getAllCalendarsFeed(feedUri, callback, handleError);
```

The code snippet here leaves out some details. Please consult the Google calendar for full working code [ref].

4. Finally, we create a new entry in the user honeydew's calendar using the following codes:

```
// Create an instance of CalendarEventEntry representing the new event

var entry = new google.gdata.calendar.CalendarEventEntry();

// Set the title of the event

entry.setTitle(google.gdata.Text.create(<eventName>));

// Create a When object that will be attached to the event

var when = new google.gdata.When();

// Set the start and end time of the When object

var startTime = google.gdata.DateTime.fromIso8601(<startDate> + "T" +
<meetingStartTime> + "-05:00");

var endTime = google.gdata.DateTime.fromIso8601(<endDate> + "T" +
<meetingEndTime> + "-05:00");

when.setStartTime(startTime);

when.setEndTime(endTime);

// Add the When object to the event

entry.addTime(when);

...

// Submit the request using the calendar service object

calendarService.insertEntry(feedUri, entry, callback, handleError,
google.gdata.calendar.CalendarEventEntry);
```