Developing of the PALMSCom Widget

By Egil Hansen, egilhansen.com, Fall 2011

Introduction

This report captures the process of developing the requirements for the PALMSCom widget, from the early beginnings where the target audience of the widget was considered, all the way to a functional prototype presented to stakeholders.

Most of the content is based on the notes I took during the project, with some added reflections, updates, and corrections, where appropriate.

Background

The PALMS system is web-based system used by exposure biology researchers to analyze and share datasets. If a PALMS user needs help, they communicate directly with the PALMS support team, which creates a vertical communication channel that other PALMS users cannot see or contribute to.

PALMSCom is supposed to solve that issue by allowing the geographically divided users to communicate between each other, to allow them to build a community where they can share discoveries and support each other, as well as talk directly to online supporters if they run into problems, all in real time.

The goal of the PALMSCOM widget is to enable informal, real time, communication between users that is low-cost in terms of required engagement from the individual user. It is important that the widget does not distract users from their primary purpose for using PALMS, so the widget must exist as a secondary item and not get in the users way when they don't want it to.

Table of Contents

The Audience

The primary users of the PALMS system are exposure biology researchers who are well educated and can for the most part be considered super users. They use computers on a day to day basis using specialized software to do their job, they know the general concepts and conventions used in software, e.g. the place to look for "Print" command, so it is reasonable to assume that they are able to start using a new piece of software at an adequate right away, as long as such software adhere to de-facto standards in computer software.

The secondary users of the PALMS system are the PALMS administrators, supporters, and developers. Their level of computer expertise should allow them to pick up any new functionality added to PALMS without any problems.

Conversation Stream

The conversation stream is the stream of real time messages sent between users, the core part of the PALMSCom widget. In this section we look at what it is and equally important – what it is not.

What it is not

The conversation stream is not meant to enable:

- Long conversations
- Discussion of ideas in details or length
- Function as a mailbox
- A mean for private communication (private messages)

Most chat systems allow users to send private messages to each other, so why not here? Since this widget will operate with a single conversation stream, it seems unnatural to embed private messages into that stream. It also goes against the idea of community building, and will enable the vertical communication that exists today between users and admins. The belief is that if users want to have a private conversation, they use other types of communication such as email or Skype.

Conversation Stream properties

Messages are broadcasted to the conversation stream, which is visible to all online users. New users logging in will see the conversation stream containing the latest messages, and it is also possible to scroll to the bottom of the available conversation stream and request more (older) messages to be loaded.

Messages should not be allowed to be very long, somewhere between 300-400 characters, but feedback from users can change this. Longer messages will not fit well with the presentation of the conversation stream, which is likely to be a narrow column in the right or left side of the screen, and the type of conversation the widget is supposed to support do not require long messages.

The question is if it is necessary to impose a limit at all.

Replies

It is possible to respond to another user by including their name in a new message. Such responds are considered important so they will be highlighted in the receiver's widget.

Keywords

It is possible for a user to specify certain words (keywords) that he/she wants to get notified on, when a message containing them appears in the conversation stream. This was a specific request from a stakeholder.

Attachments

It is possible to add file attachments such as images, videos, etc. to a message. These will be stored on the widgets server and a link to the attachments will be displayed with the message.

Alternatively, certain content types such as images or videos could have a thumbnail version displayed alongside the message. However, I believe this could lead to unnecessary noise in the conversation stream. It would probably be better to just display images or videos in an unobtrusive popup when a link to them is clicked.

Comparison of conversation styles in existing chat technologies

For inspiration, I have taken a look at existing chat technologies and give a crude analysis of how their characteristics.

	IRC/Twitter/Facebook	**Message Boards/Mailing Lists/Google+**
Message length	Short (160 characters to ~500 characters)	Long
Message type	Very informal	Depends, can be very formal
Engagement	Low	Medium to high
Message backlog	Not important	Depends, can be important
Time to post	Short	Medium to high
Conversation type	Water cooler style, short questions	Longer discussions, detailed questions
Communication	Instant	Delayed, spread over time

Table 1: Comparison of conversation styles in existing chat technologies.

Initial Requirements and Paper Prototypes

With the above in mind, the initial requirements for the widget are:

- The widget should be collapsible/sizeable. For users with little screen real state and users who would prefer less noise.
- The widget should have an indicator for drawing attention to replies, tags. This could be in the form of a blinking icon or adding a blinking effect to the message that requires attention.
- The widget should show an hideable list online users.
- There should be a button linking to a settings view.
- There should be a button linking to a search view.
- There should be a dedicated search view.
- The conversation stream has its own internal scrollbar.
- The widget has the full height of browser window.

Update: We wanted to bring several different prototypes to the first stakeholder meeting and came up with four different collapsed views and one expanded view. Figure 1 shows the expanded view and all the other prototypes can be found in the appendix.

Figure 1: Paper prototype, expanded view.

General observations about the views

Since the existing PALMS user interface is very subtle already, no extreme notification effects are necessary to gain users attention.

It might even be enough notification for users just to see a new message appear (in the expanded view at least) to notice it and thus participate. This assumes that there is not a constant stream of messages, in which case there is already a conversation ongoing (or somebody is spamming). I assume that there normally is little activity in the conversation stream, which means just the appearance of a new message, should be enough to trigger users' peripheral vision, even if they only see a truncated version of a new message in the vertical view/horizontal view.

Serves user \ View	**Vertical view**	**Horizontal view**	**Boxes view**	**Triangles view**
Responding	n/a (1)	n/a (1)	n/a (1)	n/a (1)
Notification	Can use a blinking effect, etc. (1)	Can use a blinking effect, etc. (1)	Can use a blinking effect, etc. (1)	Can use a blinking effect, etc. (2)

Being Interested	Displays the latest message, truncated to fit.	Displays the latest message, truncated to fit.	New message counter (3)	New message counter (3)
Participating Level	Limited. Users only get to see the latest message truncated, with text going from top to bottom, possibly making it harder to read quickly. It is possible for users to glance and read at least some parts of message and decide if they want to see more or respond.	Limited. Users only get to see the latest message very truncated. It is possible for users to quickly glance and read at least some parts of message and decide if they want to see more or respond.	Low. (4)	Low. (4)

Table 2: Comparison of how the different collapsed views serves the user.
(1) User cannot respond in this view, requires two clicks to start a response, one two expand to the Expanded view, and one to trigger the input box.
(2) Flash expand icon in a different color than default, or flash entire view, even play a "ding" sound.
(3) The message counter has a limited to it; it will stop fitting when there are more than 2-3 digest.
(4) Users only have a number indicator telling them about new messages. They have to actively move mouse cursor, click on box to see messages and then decide if they want to participate.

Peripheral Attention and Notification

In this section I discuss how the knowledge about peripheral attention can help us create a widget that will not disturb users in the normal work with PALMS, but still draw their attention when something of interest is going on.

Abstraction

The key to make peripheral attention work is to represent information in an abstract form. This will allow the user to use his/her peripheral vision to keep track of changes. *Abstraction involves extracting features or reducing the fidelity of information so that it is easier to read "at a glance" than the raw input. Abstraction enables lower attention consumption of information.* [1]

In our case, where changes are in the form of new messages being posted to the conversation stream, what needs to be abstracted is:

- Amount of messages being posted per minute (or other fitting time unit)
- The specific content of a message, e.g. keywords (tags), user names
- Author of message

The timestamp of a message is not something I will consider as relevant information for peripheral consummation, since a change occurs when a new message has been posted and the time that message was posted is thus known implicitly.

The author of the message might not be relevant, unless PALMS wants to enable users to highlight message from certain users, or want to enforce certain messages from certain users to be considered higher value (e.g. admins).

Activity states

With the above in mind, the following three activity states come to mind:

- No activity (all online users are inactive, or no users are online)
- Activity
- Activity, high importance to current user

These states can be mapped to the three notification levels defined in [1]:

- Ignore
- Change blind
- Make aware

When there are no activity, there is also no need to engage the peripheral vision of the user, i.e. the ignore notification level.

When there is activity, but not activity that the user has deemed as being of high importance, the *change blind* notification level seems appropriate. The *change blind* notification level is defined as representing information of marginal importance that should be displayed in a way that consumes no conscious awareness but may still effects behavior, corresponding to *inattention* defined in [1].

When there is activity that the users considers important, the *make aware* state is fitting. At this notification level, information is represented in such a way that it consumes some attention of the user, corresponding to *divided attention* defined in [1].

I don't think we should consider notification levels that require the user to act or interrupt their work. That seems to be crossing a line where we start to force the user to take part in the community.

This actually leaves us with five different notification levels, since the widget will be available in at least two states (collapsed, expanded). *Change blind* and *make aware* will use different types of transitions depending of the state of the widget, effectively allowing the user to also control how invasive the widget will be in terms of attention grabbing.

Transitions (attention grabbers)

In [2], they show that slow fading, changing of colors, is well suited for conveying *change blind* state. Users will register a new state via their peripheral vision as through change in colors or in new text from a message being faded in or added in subtle way to the conversation stream.

More abrupt transitions are needed for the *make aware* notification level. This could for example be adding a blinking effect to messages that fall into this category, or having the animated in a slow moving way to make them stand out.

Changing notification states

Each widget view (expanded/collapsed) has, or share, four notification states. The states represent how the view will change it appearance to grab the users' attention, in various degrees, depending on the state.

How the appearance will actually manifest is not included in this state diagram, as it is not relevant in this context

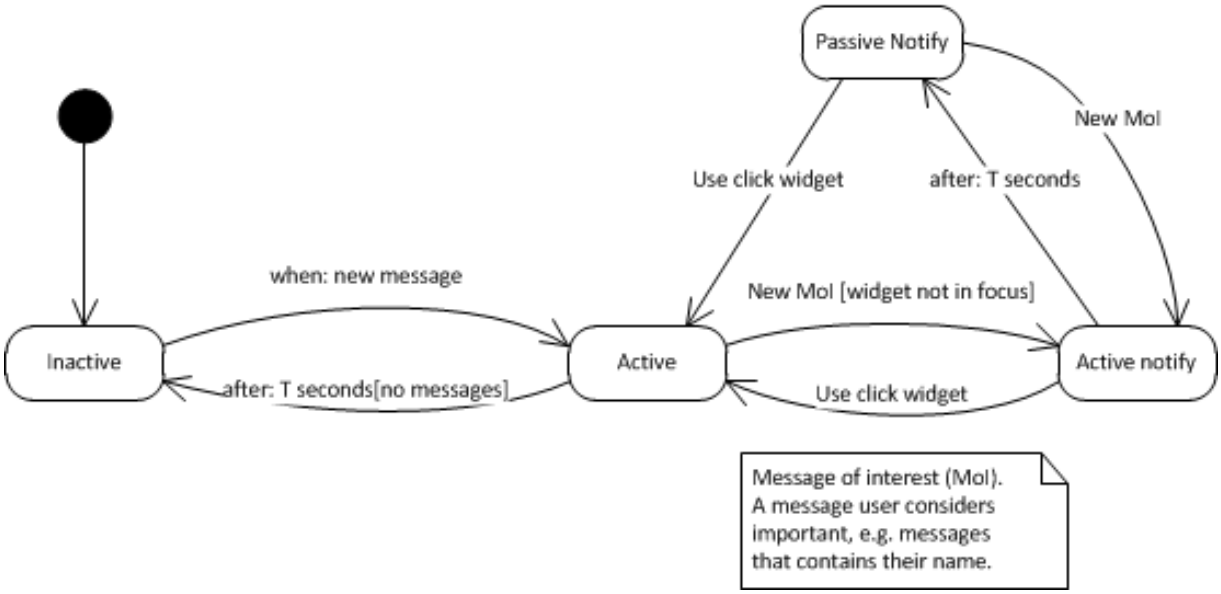

Figure 2: State diagram showing the different notification states and transitions between them.

Notes about Active and Passive notify states

The difference between "Active notify" and "Passive notify" is that active will be the most "noisiest", but will only be triggered for a short time interval. That gives the user the option to ignore the notification without having to turn it off.

The "Passive notify" state is meant to support scenarios where users are not at their system when the "Active notify" is triggered. That why they do not miss a notification, even after the active notification has timed out.

General Notes

If the user already has the widget selected, e.g. the input field, the users attention is already on the widget thus mitigating the need to change to a notify state.

We go to an inactive state if there have been no new messages for a specified amount of time.

User Stories

Id	Story	Done
1	As a *PALMS user*, I want *to sent messages to other users*, so that *I can share my experience and get help when I am stuck.*	X
2	As a *PALMS user*, I want *to be able to reply to another user*, so that *the user can see that I am talking back to them.*	X
3	As a *PALMS user*, I want *to be able to attach a screenshot of my screen to a message*, so that *it is easier for other users to help me solve my problem.*	
4	As a *PALMS user*, I want *to be notified when somebody replies to me,* so that *I do not miss a message sent to me.*	X
5	*As a PALMS user,* I want *to be notified when somebody users a keyword I have defined in a message*, so that *I do not miss conversations I am interested in.*	X
6	As a *PALMS user,* I want *to be able to define keywords I find interesting,* so that *I can be notified when a message of interest is posted.*	
7	As a *PALMS user,* I want *to be able to view screenshots attached to messages,* so that *I can better help.*	
8	As a *PALMS administrator*, I want to *capture and log conversations for offline analysis.*	
9	As a *PALMS user*, I want to *be able to scan or search previous message*s.	
10	As a *PALMS user*, I want to *be able to look back into the message list to see what has happened*.	
11	As a *PALMS user*, I want to *be able to see who is online and their skill level,* so that *I can see if there is any people online to talk to or get help from*.	X

Lessons Learned

The lessons learned during this project:

Message length: As discussed in the conversation stream section, limiting the length of a message was not relevant. We did not end up setting any implementation in the prototype and it seems like the UI and format of a message is such that there is a natural limit to the length of messages.

No need to complicate replies: My original plan was to use Twitter's convention when replying to another user in public, i.e. using the @ symbol as shown in the first paper prototypes, but after receiving feedback form our stakeholders, I dropped that idea in favor of the more natural IRC style, where including an users name in a message is enough.

No need to complicate messages: I originally planned to let sending users explicitly tag messages using the #-notation, e.g. #help. The purpose was to allow metadata to be attached to messages, making them easier to search. However, stakeholders once again pointed out that unless you know the Twitter convention it was not intuitive.

A list of online users: A stakeholder suggested the obvious addition of a list of online users, and to have that list include which category each user was in (e.g. admin, supporter, user), which turned out to be a core feature that was a must have.

Auto load older messages automatically: It was also suggested by a stakeholder to automatically load older messages if the user scrolled down to the bottom of the message stream, instead of forcing the user to manually click a "Load more messages" button

Transitions work: The feedback from the stakeholders so far suggests that the chosen notification levels and transitions are appropriate (as described in the section Peripheral Attention and Notification).

On a personal note, I was once again reminded that when developing prototypes, I need to focus on features that can move the entire process forward, and not get stuck for days trying to make the communication between server and clients go 1ms faster.

Implementation

In its current incarnation, the PALMSCom widget prototype implements the features listed as done in the Users Stories table above. There are two states, a collapsed and expanded, and two different collapsed views, the horizontal and the vertical. The widget can both float on top of PALMS and overlay it if expanded, or float beside it.

Code Design Patterns

Besides the observer pattern, which is heavily used throughout the widget, the following patterns are used more or less extensively:

MVP Pattern

The widget is build according to the MVP pattern, as recommended by Google[1]. This makes it possible to unittest most of the logic code (although no unittests is written at this time), and it should be fairly straight forward for anybody familiar with the MVP pattern to extend the widget.

Affected classes:

- AppController (sets up connection to the web service, performs initial login, and handles changes in state, i.e. switch between collapsed and expanded view)
- ExpandedPresenter
- CollapsedPresenter

[1] Article from Google describing how to architecture a GWT application following the MVP pattern: http://code.google.com/webtoolkit/articles/mvp-architecture.html

- ExpandedView
- CollapsedView

Decorator Pattern

The Decorator pattern is used to decorate the message class on the client. There is extra functionality and processing needed on the client that is irrelevant on the server, but naturally belongs in the message class, so the Decorator pattern enables keeping unnecessary complexity out of the message object transmitted between the client and server while keeping things neat on the client as well.

Affected classes:

- Message (Interface)
- MessageImpl (implements Message)
- MessageDecorator (extends MessageImpl, implements Message)
- ClientMessageDecorator (extends MessageDecorator, only used on the client)

Proxy pattern

The proxy pattern is used to abstract the client/server communication away on the client, so the client/server communication implementation can be replaced in a later phase of the project with a more efficient.

Affected classes:

- PalmscomService (interface, describes the RPC methods)
- ServiceProxy (client proxy, abstract class)
- PollingServiceProxy (extends ServiceProxy, implements automatic polling of new messages from server)
- InMemoryPollingServiceImpl (implements PalmscomService, stores data in memory)

Singleton pattern

There is a crosscutting issue in the PALMSCom widget, client application state, the current user and user settings, to be exact. There are numerous ways to deal with this, one way to handle it is dependency injection, but to keep things simple I choose to go with the Singleton pattern and a single AppState class.

Future Work

The following features are those that should be considered for the next version of the widget. They were the most mentioned during meetings with stakeholders that did not make it into the current version.

- Integrate a screenshot feature into the widget.
- Update the look and feel of the widgets UI to match that of PALMS more closely.
- Allow the user to opt-out, i.e. set the widget in a do-not-disturb.
- Allow admins or users to read through old stored message logs.

Integration into PALMS

In this section I will outline how PALMSCom can be integrated into PALMS. In short:

On the client

- Add *Palmscom.css* to client project
- Create an instance of the *Palmscom* class, pass in the current user object

Communication (client/server communication)

- Replace current RPC based message polling with COMET based pushing of new messages

On the server

- Save user settings to repository (e.g. database)
- Save messages to repository (e.g. database)

Add Palmscom.css to client project

The Palmscom.css file contains the custom formatting for the widget that makes sure that the widget is correctly overlaid PALMS and behaves as expected. Some of the styling rules can be changed without a problem, e.g. typography rules and colors.

To make the overlaying work probably, it might be necessary to add "*z-index: 1"* to the main PALMS container, to make the overlaying work unobtrusively. See the demo application for details (*Palms.css*).

Replace current RPC based message polling with COMET based pushing of new messages

For production use, it is almost certainly required to implement COMET based push communication between client and server. In the current mockup client/server implementation, I am just constantly polling the server through RPC to achieve almost realtime communication. This is clearly a waste of resources, so utilizing a library COMET library like socket.io[2] to keep communication realtime and avoid continuous polling is the obvious choice.

Since the client abstracts away the library used to communication with the server through the *ServiceProxy* abstract class, the client/server communication code can easily be replaced.

For a good starting point, I recommend taking a look at the *InMemoryPollingServiceImpl* (server) and *PollingServiceProxy* (client) classes.

[2] Socket.IO (http://socket.io) aims to make realtime apps possible in every browser and mobile device, blurring the differences between the different transport mechanisms. It's care-free realtime 100% in JavaScript. A promising looking Java backend integration for Socket.IO library is available over at GitHub: https://github.com/Ovea/Socket.IO-Java

Known Issues

- The vertical collapsed view's blink background color is not rotated with the text, does not go all the way down to the bottom of the screen. Seems to be a limitation in CSS3's transform: rotate() function.
- Online users list does not always expand enough to show all users. Probably a CSS issue.

References

[1] T. Matthews, A. K. Dey, J. Mankoff, S. Carter and T. Rattenbury, "A Toolkit for Managing User Attention in Peripheral Displays," in *UIST'04*, Santa Fe, New Mexico, USA, 2004.

[2] D. S. McCrickard, R. Catrambone and J. T. Stasko, "Evaluating Animation in the Periphery as a Mechanism for Maintaining Awareness," 2001.

Appendix

Paper prototypes

Figure 3: Expanded view

Figure 4: Expanded view, with input box active.

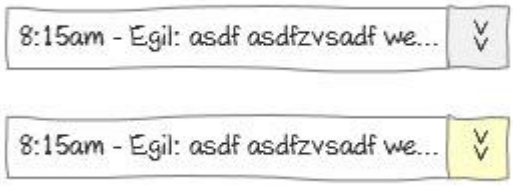

Figure 5: Collapsed view, horizontal. Top shows normal mode, bottom shows notification mode.

Figure 6: Collapsed view, box. Top shows normal mode, bottom shows notification mode.

Figure 7: Collapsed view, triangle. Top shows normal mode, bottom shows notification mode.

Figure 8: Collapsed view, vertical. Left shows normal mode, right shows notification mode.

Screenshots of the PALMSCom widget

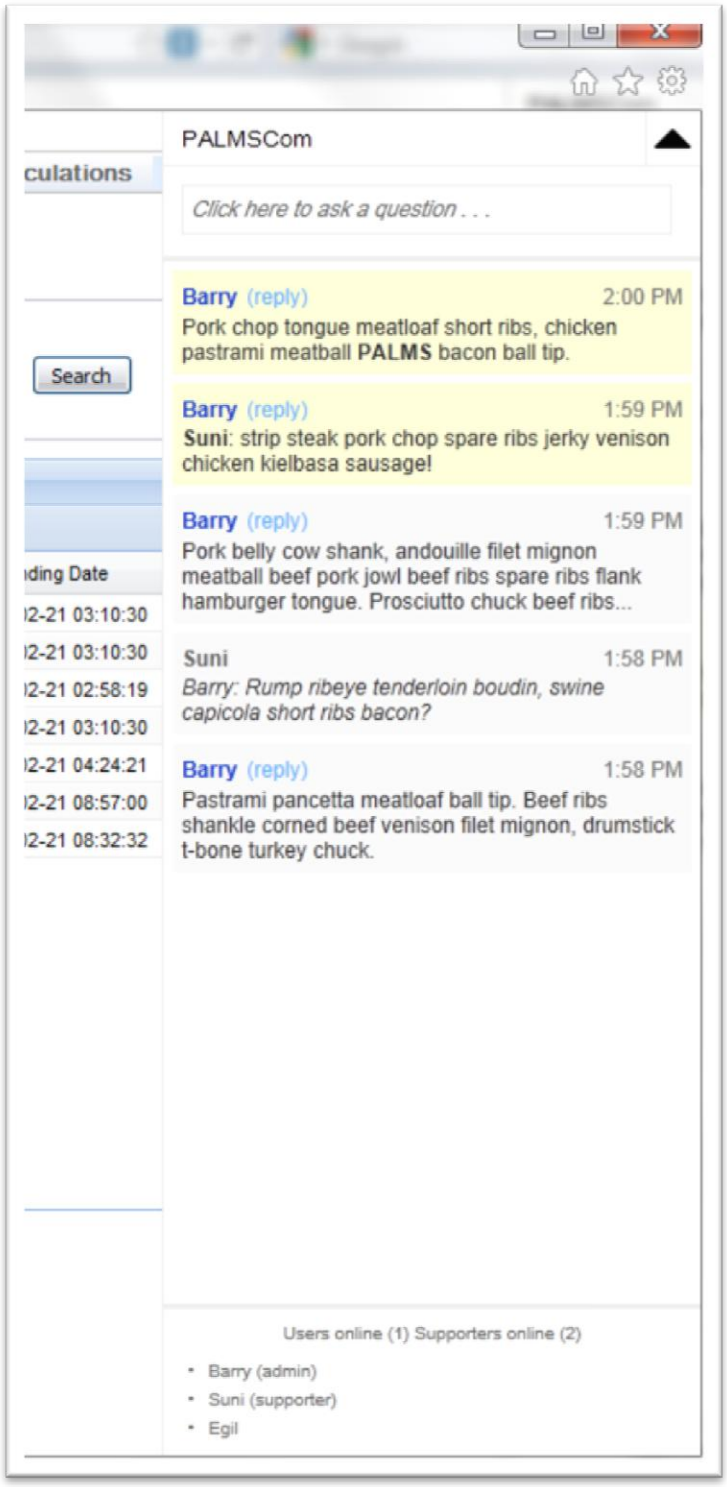

Figure 9: PALMSCom in expanded mode. Messages with yellow background is "Messages of Interest" to the currently logged in user. Messages in italic are the current users own messages.

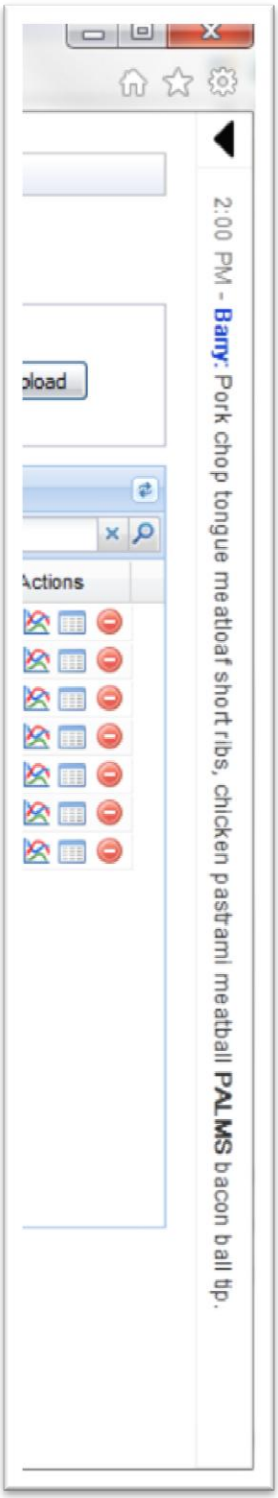

Figure 10: PALMSCom in vertical collapsed view.

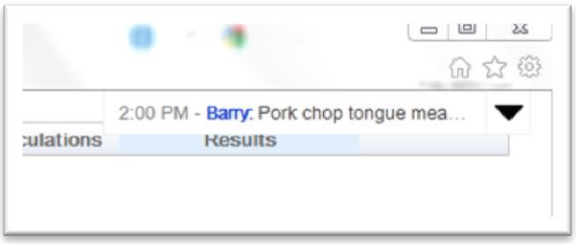

Figure 11: PALMSCom in horizontal collapsed view.

Figure 12: The online users list in expanded mode.