Groupware Plug-ins: A Case Study of Extending Collaboration Functionality through Media Items

Gregor McEwan, Saul Greenberg, Michael Rounding and Michael Boyle
Department of Computer Science
University of Calgary
[mcewan, saul]@cpsc.ucalgary.ca

Abstract

Groupware normally offers only fixed functionality, which can be a poor match to the actual needs of particular group. We argue that groupware should be extensible by third party developers, and describe groupware plug-ins as a method that enables this. Using the Community Bar (CB) as a case study, we illustrate an easy-to-program extensible groupware architecture. Unlike single user plug-ins, CB groupware plug-ins automatically share and populate a distributed data structure, using a distributed Model View Controller pattern to simplify programming. Several third party plugins illustrate what people can create in practice.

1. Introduction

Single-user applications typically provide users with fixed capabilities. Offerings range from limited feature programs focused on task specificity (e.g., a photo viewer), to a smorgasbord of functions included to appease a broad set of users performing quite different tasks (e.g., *bloatware* such as popular word processors). Groupware is often designed the same way, where designers which capabilities are in and which are out. Yet groupware is more than a productivity tool, for it also includes communication and information sharing. The problem is that, except for the most generic of collaborative activities, the offerings of the groupware designer will likely be a poor match to the actual needs of the group as overly complicated bloatware.

One approach to solving this problem is to create groupware as an extensible architecture. While there are many ways that this could be achieved, our particular interest is in *plug-ins,* a method popular in extensible single user systems. Plug-ins are usually implemented as visual components that can be incorporated at runtime within some kind of graphical user interface container. Each plug-in delivers custom - albeit limited - capabilities to its users. For example, plug-in 'tickets' in Sideshow [2] range from traffic reports to stock tickers, while Google Sidebar plug-ins [http://desktop.google.com] may show news listings, weather, a musical play list, and so on.

Plug-in systems are comprised of:

- a base system that serves as a (visual) container of components;
- a stock set of plug-ins that the system designer supplies to anticipate basic user needs;
- a development kit to allow third party designers to create custom plug-ins that are automatically included without recompilation into the system.

In this paper, we describe how plug-ins can also work as part of an extensible groupware architecture through one vital addition:

- plug-ins can become groupware-enabled by automatically sharing and populating a distributed data structure that includes simple, complex and multimedia data types, and that uses a distributed Model View Controller pattern to simplify programming.

To show how this works, we detail our Community Bar groupware system (CB) as a case study of an extensible networked architecture. To set the scene, we first summarize the CB interface, followed by its architectural requirements and details. We then describe how third party programmers can easily create and add new groupware functionality via custom groupware plug-ins called *media items* [7]. We close with a few third party plugins that illustrate what people can create in practice.

2. Community Bar

Community Bar (CB) is groupware intended to support causal interaction within a small distributed group. CB presents itself as a sidebar peripheral display [2] – a space-conservative bar on the side of the screen that can never be covered by other applications. Figure 1 illustrates its appearance, with its primary features summarized below. While we provide an overview here, CB's design rationale and interface is described fully in [5,6].

Places. Using a menu (not shown), people create, name and/or join 'Places'; each Place serves as a

locale [3] collecting different people and other shared information. Individuals can belong to different groups, and thus each can have different places visible in their Community Bar. For example, Figure 1 shows an individual's view of three on-line places, titled: *CSCW class*, *ilab*, and *mike test.*

Media items. Each place contains a number of *media items* [4,7]. These comprise tiles, tooltip grandes, and full views as described below.

Tiles. The sidebar view of each place contains a number of *tile* views of several stock media items included in CB. As illustrated in Figure 1, these small tiles can represent things like people (as live video, photos or names), public conversations (as public chat dialogues or sticky notes), or publicly shared information (e.g., web pages of common interest).

Tooltip Grande. Tiles typically provide only basic awareness information. Individuals can choose to explore and even interact with that information in greater detail. When a person mouses over a tile, CB will display a *tooltip grande* next to it [2]. For example, Figure 1 (left side) illustrates the tooltip grande for the Presence tile. While the tile shows a low fidelity and infrequently updated video image, its tooltip grande contains a higher fidelity and more frequently updated image as well as various controls.

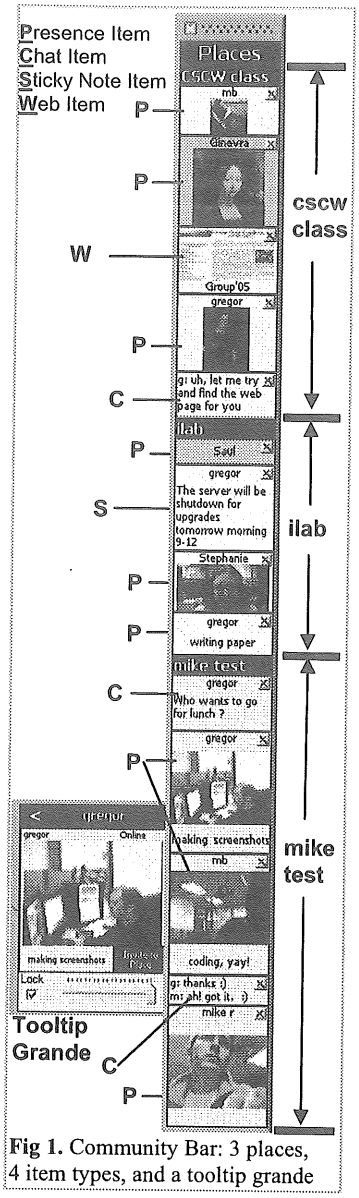

Fig 1. Community Bar: 3 places, 4 item types, and a tooltip grande

Each tooltip grande also contains a 'focus' slider control that allows the user to control one's personal view of tiles, i.e., the tile can be made smaller, which may semantically alter the information displayed within it so that it is appropriate to its reduced size.

Full View. When a person clicks on the title bar of the tooltip grande, a new separate window called the full view displays even richer information, and makes available all the functional capabilities of the item. For example, the full views of Figure 1's Presence items is seen in Figure 3, where it contain even higher resolution and higher frame rate video, a static picture (in case the person is out of view), and the ability to enter into a vocal conversation through a 'Push to Talk' button.

Owner vs. Audience. An *owner* is the person who originally creates and posts a media item, while the *audience* are all others who can view it. The designer of a media item can have its tile, tooltip grande or full view behave differently for the owner versus the audience. For example, the Presence item offers the owner additional control on whether the audience should view one's video or a static photo (Figure 3)

Group control. Places, the membership of people to that place, the choice of media items within them, and the content of these items are completely defined by the group on a moment by moment basis. While people in a particular place may see similar things, individuals can control the size of tiles within it, and separately drill down for further information through the tooltip grande or full view.

From Awareness to Interaction. For each media item, its tile view generally shows awareness information; its tooltip grande shows more detailed information and allows partial interaction; while the full view shows all the information and all communication and interaction possibilities. For example, in the Chat Items of Figure 1, the tile view shows the last message or two, the tooltip grande view shows the last 10 messages and allows sending brief messages, and the full view allows sending long messages while showing all messages, the place members, and who is currently typing. Of special note is the full view of a Place, which fits all the tooltip grande views of a place's media items into a window as a rectangular grid (not shown). In this manner, the full view of a Place almost completely implements, and therefore subsumes, all capabilities of the Notification Collage [4,7].

3. Architecture

The Community Bar architecture has three major components. The first is the main application, including the *Sidebar Container* interface and management of People and Places. The second is the *Media Items Container*, which wraps each plug-in created by developers and presents it within a Place. The third is the *Shared Dictionary Notification Server* that forms the underlying distributed system.

This architecture was designed around four key requirements:

1. Networking and Distribution – it must be accessible to distributed groups of people;
2. Universality – a standardised Data Model that represents people and their content;
3. Application – a program that manages and displays people and their content; and
4. Extensibility – allow average programmers to easily add new custom content.

In the following subsections, we describe how the architecture is applied to the first three requirements. Section 4 then describes Media Items, which fulfill the fourth requirement.

3.1. Networking and Distribution

Because CB supports distributed groups and people, CB's first requirement is that it must have a distributed architecture that manages multiple processes across different computers, and the ability to share and transmit data between them. This implies quite complex networking and data distribution requirements that could easily become a programming nightmare in an extensible plug-in architecture.

To manage this complexity, we built CB as a client/server architecture whose data distribution is based upon a distributed model-view-controller (dMVC) pattern combined with a notification engine. The basic ideas are:

- ***Model***: the system maintains a persistent data store on a server that is accessible to all clients,
- ***Controller:*** the model is updated by distributed clients, usually as a result of user actions,
- ***Views:*** client interfaces will be updated to reflect the current state of the model.
- ***Notifications:*** the server will generate notifications to all distributed processes about additions, changes and deletions to that data.

To implement this pattern, CB uses GroupLab.Networking [1], a third generation publicly-available networking engine and toolkit developed within our laboratories. It proved a good fit for the networking layer of Community Bar for several reasons. First, it provides a server with a persistent data store via a *shared dictionary*: a hierarchical key/value attribute list with an easy-to-use API for setting and retrieving distributed data. Second, the engine automatically updates clients via a publish/subscribe *notification system*, where clients subscribe to data by pattern-matching particular hierarchical branches and nodes of the shared dictionary. Thus the dMVC / notification pattern is easily implemented. Third, all low level networking details are hidden, as they are automatically managed by the .Networking runtime architecture. Finally, it automatically marshals and stores data ranging from simple data types (strings, integers...), complex data (structures, lists...), and multimedia (binary data, images...).

3.2. Universality (Data Model)

Community Bar needs to maintain a strong notion of three entities: of places, of people, and of media items. All three entities are potentially long-lived and information about them is needed by each client. Thus CB's second requirement is to have persistent and distributed representations of places, people and items.

To achieve this, CB uses the .Networking shared dictionary as a server to persistently store all the data about places, people and items. To illustrate how this is done, we will use the *Meeting Room* place illustrated in Figure 2 (screen capture on the right side), which is currently inhabited by two people and which displays two presence and one chat media item. Figure 2 also shows the (slightly stylized) corresponding data model held in the shared dictionary server that defines the people, places and items in the *Meeting Room* place.

CB leverages the hierarchical nature of the shared dictionary by separating data into well-known branches of the data tree. The top-level *users branch* stores data about a person, e.g., email, initials, online/offline state, and so on. Figure 2, for example,

Key	Value
Users	
/users/judy@mail	**Judy**
/users/ judy@mail/info	initials = JO
/users/saul@mail	**Saul**
/users/saul@mail /info	initials = SG
Places	
/places/guid1/	**MeetingRoom**
/places/guid1/member/guid2	/users/judy@mail
/places/guid1/member/guid3	/users/saul@mail
Media items	
/places/guid1/items/guid4/type	**Presence**
/places/guid1/items/guid4/owner	/users/judy@mail
/places/guid1/items/guid4/place	/places/guid1/
/places/guid1/items/guid4/picture	<picture data>
/places/guid1/items/guid5/type	**Presence**
/places/guid1/items/guid5/owner	/users/saul@mail
/places/guid1/items/guid5/place	/places/guid1/
/places/guid1/items/guid5/picture	<picture data>
/places/guid1/items/guid6/type	**Chat**
/places/guid1/items/guid6/owner	/users/judy@mail
/places/guid1/items/guid6/place	/places/guid1
/places/guid1/items/guid6/msg/	sender=Judy; initials=SS; message="Want..." sender=Saul; intials=SG; message="Sure..."

Fig 2. Sample place and its Shared Dictionary key structure

shows how information about its two people is stored in the model's users branch, where each person's subtree is defined by their unique email address. Next, the top-level *places branch* stores information about places created by a sub-group. It also contains a list of all members of the group, which in turn points back to particular people in the people branch. For example, the single place in Figure 2 is identified by a globally unique ID (the GUID), and contains information about its name and the two people within it. Finally, *media item branches* contain generic information common to all media items, as well as any local data particular to that media item type. As illustrated by the three media items in Figure 2, all items have a type, an owner, and the place it resides in (which points back to the places branch). Other data is specific to its function (e.g., the multimedia picture item, or the complex data structure comprising the actual text chat); Figure 2 shows this in italics. Because items can only exist within a place, they are stored as sub-trees within a particular place branch.

3.3. The Application

The CB Sidebar Container client structures people and places, and servers as a container for the Media Item plug-ins. It performs this role with respect to both interactions with the data model and as a visual container in the User Interface (UI). Media Items are contained within Places, and People subscribe to places to view the Media Items. The CB Sidebar Container therefore manages the data and user interface aspects of Places, People and the generic aspects of Media Items.

Place and People data, shown at the top of the table in Fig. 2, is controlled by the CB Sidebar Container. It gathers and updates all the People data (e.g., display name and initials), the Place data that People are subscribed to, and which Media Items are in the Place. The Sidebar also manages the general Media Item data in the fields *type*, *owner* and *place* (shown in Fig. 2). All of the other Media Item data (in italics in Fig. 2) is entirely managed by the Media Item itself, as discussed in Section 4.

The Sidebar Container currently implements a generic sidebar interface [2], managing placement of controls and displaying the tooltip grande. It acts as a visual container for Media Item visual components. Place headers are displayed in the bar (for example in Fig. 1 there are three Place headers shown), and a space is reserved on the bar for each of the Media Items within each Place. The Sidebar Container queries each Item for a Tile view, which is then displayed in the reserved space. Similarly, it also deals with the Tooltip Grande, displaying it when the user moves their mouse over the bar. Media Items are queried for the visuals of this view, which is then placed in the Tooltip Grande space.

In summary, the Sidebar Container links the shared data model, the UI and Media Items by:

- querying the media items for controls to place in the bar and tooltip grande;
- passing user commands to the UI to send to the shared dictionary; and
- responding to notifications from the data model to appropriately update the UI.

4. Media items

Our final and most important requirement is that the Community Bar should be extensible by average programmers. We strongly believe that third party developers should be able to create media item plug-ins without excessive training and effort.

To satisfy this requirement, the CB Sidebar Container was designed as a groupware that manages an ad hoc collection of media item plug-ins, whilst managing People and Places and providing networking and shared data facilities (see Section 3). In contrast, the media items provide the actual awareness and interaction content. CB includes a basic set of generic media items, but encourages third party developers to create more specific Media Items through its plug-in capabilities.

4.1 The Plug-In

Media item plug-ins are based on an easy-to-learn development platform offering a relatively simple object-based programming metaphor, easy access to the distributed data model, and a development environment for rapid testing of ideas. The programmer creates and compiles media items as individual Dynamic Link Libraries (DLLs). CB then loads these DLLs dynamically at runtime; programmers do not need to access or recompile CB source. Most of the programming effort is on the custom functionality of individual media items, e.g., to implement a chat system and its interface. The only extra effort required is that programmers of these DLLs have to implement a simple pre-defined code interface that the CB sidebar requires to host the item.

Perhaps the most important information given to the Media Item programmer is the handle to the shared dictionary. Unlike single user plug-ins, client groupware media items need to share data between their distributed counterparts. The dictionary, as in the main CB application, provides the model and notifications for Media Items. This allows Media Items to distribute data between item instances (belonging to the owner or one of the user audience

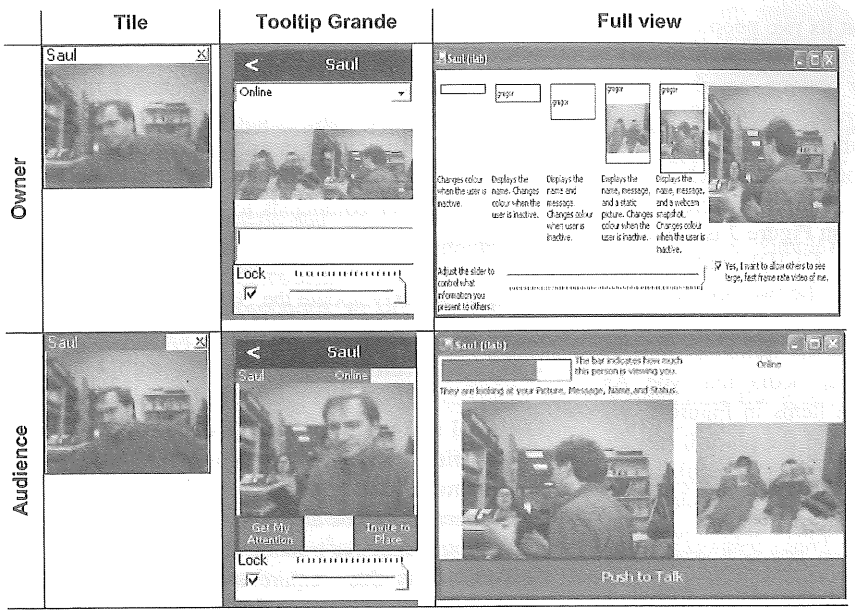

Fig. 3: Six views of the Presence Item

members), where the item act as a view and controller following the dMVC pattern.

To facilitate this, programmers are provided via a code interface with a convenient handle to the Shared Dictionary server and hence the data model. They are also given a direct reference to this media item's branch in the dictionary, and direct references to the user branch of the item's owner and of the current user. Thus programmers do not need to know or worry about the potentially complex hierarchical paths and data that are above these branches, i.e., they are exposed only to the primary data they would normally use and manipulate. Through this simplified data interface, the developers are easily able to add and distribute custom data so that all client media items display the appropriate contents for their tile view, the tooltip grande view, and the full view. However, advanced programmers can access this other data in the shared dictionary if needed. For example, a programmer may iterate through the user's branch to list all connected people.

With this access to the shared data capabilities of the CB Model, media item developers have considerable flexibility in the presentation of the items. In some Media Items (e.g. the Chat item) the Tile, Tooltip Grande and Full views are seen the same way by all users. However, the three views can differ for the owner and the audience. For example, Figure 3 shows the six different views for a Presence item. While tiles are visually similar, the owner of a tile can click on it to immediately update their video snapshot. The other views differ much more; the owner's view of the Tooltip and Full view provides control over their own appearance, while the audience's view provides opportunities to communicate with the owner. A Media Item can provide even more views. For example, a simple game may have different views for each of the players and then another view for onlookers.

4.2 Media Item Development Support Tools

We have also created a development environment (Fig. 4) so that Items could be coded and debugged on a local machine, outside of the main CB application. The environment contains two primary components.

First, a ***Media Item plug-in template*** streamlines the programming and housekeeping process of creating media items. The template serves a convenience role as it incorporates many of the setup and housekeeping tasks common to most Media

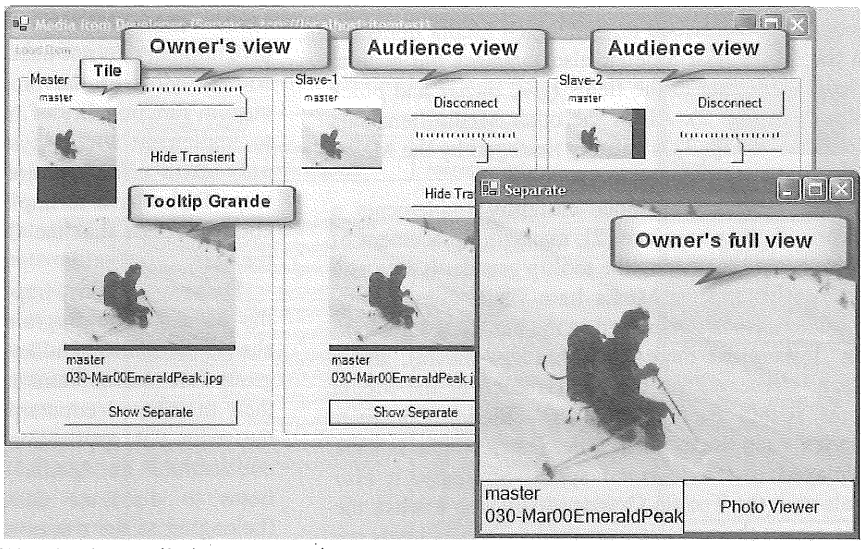

Fig. 4. The Media item test environment

Items. It also has a teaching role as it demonstrates by example the necessary components of a media item and how they relate to each other.

Second, a ***plug-in test interface*** allows a programmer to immediately test any plug-in code modifications by simulating a three-person groupware environment within a single window. As illustrated in Fig. 4, the environment conveniently mirrors CB's interaction with a media item while allowing the use of standard debugging tools. For example, the Figure illustrates a photo viewer plug-in, where any user can post photos to the group. Programmers have the option of displaying, manipulating and testing the various owner and audience versions of this photo item's tile, tooltip grande, and full view.

The simple mechanisms for creating Media Item plug-ins – an easy to use programming and data model, and a development environment – means that developers can focus on the creative and functional aspects of groupware component design rather than low level distributed systems details.

5. Examples

A class of students were asked to develop media items. Training consisted of a two hour tutorial describing the Community Bar interface and walking through an example of how to program a 'hello world' media item plug-in. Students had ~two weeks to develop and demonstrate their items. A sampling is below.

- **Public Web Item** (Stephanie Smale) displays a group editable list of web pages.
- **Video Motion Item** (Rob Diaz) provides awareness through sharing motion data from webcam video.
- **Video History Item** (Michael Nunes) displays a webcam video stream / browsable video history.
- **LCD Display Item** (Nicolai Marquardt) lets a group send messages to a physical LCD display.
- **Photo Gallery** (John McDonald): displays a group editable and browsable collection of photos.
- **Blog Reader** (Jordan Schaan) allows the group to monitor and browse a web log.
- **Scheduler** (Jeni Lynn Vito) is an event scheduler for a group's collective activities.
- **Mean Girls** (Alexandra Braginsky) lets teenage girls gossip around multimedia photos of friends.
- **Family Shopping List** (Liz Friesen) allows a family to collectively create a shopping list.
- **Digital Document Task Awareness** (Tim Au Yeung) provides task assignment and task status awareness amongst a document digitising team.
- **EBUY** (Phil Serchuk) allows the group to set up and participate in rapid real time auctions.
- **Dress Me up** (Tony Quach) supports collaborative multimedia fashion advice for an individual.
- **Bug Tracker** (Sandra Khroina) supports a team's bug tracking assignments and status.
- **Aibo Awareness** (Jim Young) allows group control of a mobile robot dog, how it views its environments, and how it interacts with people.

6. Conclusions

The Community Bar architecture serves as a case study of how groupware plug-ins can be used to extend collaboration functionality. While single user plug-ins are well known, the primary contribution of our work is to show how plug-ins can become groupware-enabled by automatically sharing and populating a distributed data structure that includes simple, complex and multimedia data types, and that uses a distributed Model View Controller pattern to simplify programming. Other contributions include the actual architectural details itself. While we do not expect future designers of groupware plug-in architectures to exactly copy our CB architecture, its basic ideas can be generalized across many different types of extensible groupware systems.

Acknowledgements. Thanks to University of Calgary students who built media items. Research is partially funded by the NECTAR NSERC grant.

10. References

[1] Boyle, M. and Greenberg, S. Rapidly Prototyping Multimedia Groupware. *Proc 11th Int'l Conf Distributed Multimedia Systems (DMS'05)*, Knowledge Systems Institute, IL, USA. 2005.

[2] Cadiz, JJ, Venolia, G.D., Jancke, G., and Gupta, A. Designing and Deploying an Information Awareness Interface. Proc ACM CSCW, 2002, 314-323.

[3] Fitzpatrick, G. *The Locales Framework: Understanding and Designing for Wicked Problems.* Kluwer Academic Publishers, 2003.

[4] Greenberg, S. and Rounding, M. The Notification Collage: Posting Information to Public and Personal Displays. Proc ACM CHI, 2001, 515-521.

[5] McEwan, G., and Greenberg, S. Supporting Social Worlds with the Community Bar. Proc ACM Group, 2005.

[6] McEwan, G., and Greenberg, S. Community Bar (The Video). Video Proc ECSCW - European Conference on Computer Supported Cooperative Work. 2005.

[7] Rounding, M. Informal Awareness and Casual Interaction with the Notification Collage. M.Sc. Thesis, University of Calgary, Alberta, Canada, 2004.