View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Servicio de Difusión de la Creación Intelectual

Un framework para la construcción de aplicaciones de comunicación textual sincrónica

LIFIA, Laboratorio de Investigaciones y Formación en Informática Avanzada
UNLP, Universidad Nacional de La Plata, Argentina

Fernando García, Diego De Sogos, Federico Naso, Ricardo Tesoriero, Alejandro Fernández
[nando,dsogos,naso,richard,casco]@sol.info.unlp.edu.ar

Resumen

Teniendo en cuenta el gran avance de las intercomunicaciones y de las herramientas informáticas existentes para tales propósitos, hay una herramienta que no evolucionó a la par del resto, el chat.

El chat es la herramienta mas popularmente utilizada para el sustento de actividades de grupo distribuida así como en el aprendizaje asistido por computadoras. Si tenemos en cuenta que cada una de estas actividades tiene sus propias características de comunicación es de esperar que se cuente con distintas variedades de chat.

Nosotros atacamos este problema construyendo un framework [2,7] que permite crear herramientas de chat de una forma fácil, sencilla, rápida y flexible. La base de este trabajo es COAST [15], un framework groupware[9] que provee independencia del mecanismo de persistencia y comunicación subyacentes.

Palabras clave

Internet, Ingeniería de Software, Groupware, Chat, Framework.

1. Introducción

La informática ha avanzado a pasos agigantados, una de las ramas que más creció, es la de las comunicaciones por intermedio de la computadora, como ejemplos podemos citar: video conferencia, Audio conferencia, Telefonía IP, etc.
Pero existe una herramienta de comunicación que no ha evolucionado a la par de las demás, el chat. Hoy en día, con una potencia de cálculo 1000 veces superior a la de los años 70, el 90% de los chats son del tipo de los desarrollados en esa década. En el caso más simple se cuenta con una ventana de texto dividida en dos secciones, donde cada usuario escribe en una de ellas[11]. Donde todos los usuarios pueden agregar su aporte en cualquier momento, las veces que lo desee y ambos pueden ver que esta escribiendo el otro, por.ej.: Talk del S.O. UNIX.
Comúnmente, las entradas se muestran en orden de llegada, e incluyen una referencia a su autor. Las aplicaciones chat más populares de la actualidad encajan en esta descripción por ej.: ICQ Chat, AOL IM, Yahoo Messager, Chatrooms, Mirc, .
El modelo general de chat es un modelo anárquico, donde cada usuario puede hacer lo que desee sin que la aplicación le imponga restricciones sobre las acciones permitidas. Por lo tanto se crea un protocolo tácito, donde los usuarios esperan a que su interlocutor escriba para luego responderle. Este protocolo es más difícil de respetar en chats grupales, donde sería necesario coordinar explícitamente la comunicación.
Otra posibilidad es tener un moderador, por ejemplo si deseamos modelar un debate entre políticos, donde uno de las dos personas tiene la palabra y luego de hablar, se calla y escucha al otro, o al moderador. La tarea del moderador es dirigir el debate, a la gente que esta escuchando o cerrar ideas de la exposición realizada previamente. Muchos modelos de comunicación tienen protocolos específicos (por ej.: Brainstorming, Debates, Enseñanza, etc.) los cuales no son modelados ni soportados por las herramientas existentes.
Debe también notarse que el modelo actual de este tipo de herramientas se limita a considerar un aporte como caracteres individuales o a lo sumo líneas de texto marcadas probablemente con una indicación de autor y hora de creación. No se proveen, por lo general, mecanismos para aumentar los aportes combinando otros medios no textuales ni enriqueciendo su semántica.
La falta de riqueza y flexibilidad en estos aspectos trae como consecuencia, al menos, que se desconozca en gran medida la dirección que la comunicación por medio de estas herramientas debería tomar. Se ha llevado a cabo muy poca evaluación y experimentación respecto a las direcciones a seguir y su importancia.
La falta de experiencia y evolución en esta área se debe a la carencia de herramientas innovadoras y flexibles que permitan recrear los distintos contextos adecuados para cada tipo de comunicación. Si pretendemos avanzar en esta área necesitamos proveer mecanismos de construcción y adaptación de herramientas capaces de evolucionar con la velocidad y flexibilidad que el área requiere.
Nuestro objetivo es el desarrollo de una arquitectura de software reusable que simplifique y potencie el desarrollo de aplicaciones de estilo chat. Este trabajo presenta Chatblocks, un framework orientado a objetos[1,8] que permite la construcción de dicho tipo de aplicaciones en tiempo mínimo y con la flexibilidad a la que hemos hecho referencia.
Lo que se desea con Chatblocks, es realizar una arquitectura que permita la creación rápida y simple de una familia de herramientas chat. Proporcionando la funcionalidad básica de la mayoría de los chats como así también soporte para personalizar aspectos tales como la forma de crear los mensajes (si se les agrega semántica o no) y como se visualizarían.

2. Chatblocks está basado en COAST

Chatblocks esta desarrollado sobre un framework para groupware[9] sincrónico llamado COAST(**Co**Operative**A**pplication**S**ystem**T**ool)[15], el mismo provee facilidad para la creación de aplicaciones cooperativas, así como también independencia del mecanismo de persistencia y comunicación subyacentes.

Las aplicaciones cooperativas sincrónicas (groupware sincrónico) permiten a un grupo de usuarios trabajar conjuntamente y simultáneamente en datos compartidos, como un documento. El framework COAST pretende hacer el desarrollo de groupware sincrónico más fácil, con un costo comparable al desarrollo de aplicaciones monousuario equivalentes.

COAST proporciona una arquitectura de groupware básica, un ambiente para la construcción de aplicaciones groupware, y una metodología subyacente para el desarrollo de groupware sincrónico.

Los mecanismos de sincronización y replicación de los datos de la aplicación son completamente manejadas por COAST. También proporciona un espacio de datos compartido, donde pueden compartirse objetos específicos del modelo entre los usuarios localmente distribuidos. Dependiendo de la aplicación los espacios de datos compartidos pueden ir desde documentos (ej. una fracción de texto, un calendario, etc.) a espacios de información muy compleja (ej. un documento del hypermedia, un mundo virtual, etc.).

Técnicamente, hay muchas arquitecturas de distribución posibles, para realizar este espacio de datos compartido. En una arquitectura centralizada, los datos compartidos se localizan en un servidor central, y los clientes tienen que enviar solicitudes al servidor cada vez ellos quieran accederlo. Como contraposición, en una arquitectura replicada, los clientes guardan copias locales de los datos compartidos, y un mecanismo de sincronización asegura que estas copias se mantienen actualizadas. Considerando que las arquitecturas centralizadas son más fáciles de implementar, las arquitecturas replicadas pueden proporcionar acceso inmediato a los objetos compartidos sin tener que esperar demoras de la red.

3. Diseño del Framework

Antes de empezar a describir el diseño y las partes del framework es conveniente destacar cada uno de los puntos que este cubre. Toda aplicación final desarrollada con Chatblocks deberá presentar una alternativa concreta para cada uno de los siguientes aspectos:

Almacenamiento:
Repositorio de mensajes. Es decir, un almacenamiento compartido de mensajes enviados por los usuarios. Al objeto encargado de esta tarea lo llamaremos MessagePool.

Desarrollo de la conversación
Dado que COAST provee un mecanismo de permanencia de objetos y replicación, desde el punto de vista del programador no se distinguen entre estas dos situaciones, los mensajes que se envían siempre se almacenan en un repositorio compartido, y los que se reciben se toman de esta colección. Por esa razón el framework provee soporte para la creación de chat Sincrónicos o Asincrónicos.

Administración y awareness de los usuarios
Un administrador de usuarios, encargado de determinar los usuarios que tienen acceso a la conversación, que tipo de usuarios son , si poseen privilegios especiales. En Chatblocks tal administrador será un UsersManager. Puede ver “Social Activity Indicators for Groupware” [10], “Transparency and Awareness in Real-Time Groupware Systems”[12] y “Supporting Awareness in a Distributed Work Group” [13] para mayor información acerca de awareness en el desarrollo de aplicaciones groupware.

Número de Participantes
Permite que haya dos o más participantes del chat, se podría incluso limitar la cantidad de participantes a un mínimo o máximo. Esto permite la flexibilidad a la hora de crear chats con restricciones al nivel de la cantidad de participantes. Esta responsabilidad recae también en el UsersManager. Se debe tener en cuenta la diferencia entre usuarios y participantes del chat. Un participante es aquel que esta activo en la sesión de chat y participa de la misma. Un usuario, en cambio, puede no estar en esa sesión de chat en ese momento. O sea todos son usuarios pero no participantes del chat.

Roles de Usuarios
Los roles de usuarios representan los roles que cumplen los participantes del chat (Administrador, oyente, mediador, etc) dependiendo del tipo de conversación que queremos representar (brainstorming, debates, etc), debemos modelar diferentes familias de roles de usuarios.
La aplicación deberá contar con un administrador de roles, el cual será el encargado de la asignación de dichos roles a los usuarios según sea conveniente, usando Chatblocks tal administrador será un RoleManager

Formas de Recepción
Este aspecto es responsabilidad del ReceptionManager, cada usuario recibe todos los mensajes que se han enviado a lo largo de la conversación, tanto los enviados mientras estaba OnLine, como los enviados mientras no lo estaba.

Filtros
Se puede aplicar filtros a los mensajes, por ejemplo: Ver solamente los de determinado usuario o los que esten entre un rango de fecha/tiempo. Esta es una tarea a cubrir por el administrador de los mensajes enviados, el ReceptionManager.

Receptor
Los mensajes poseen un emisor, que es su autor, pero no especifican un receptor, esto es así porque con Chatblocks se desea crear aplicaciones donde cada mensaje le es enviado a cada usuario del chat, por lo tanto todos los participantes se convierten en receptor.

Criterio de Ordenación
Para cubrir este aspecto, el framework posibilita la ordenación de los mensajes de tres maneras. Este aspecto de la aplicación final le compete solo a la parte local de la misma, es decir, la aplicación de determinado filtro solo es percibido por el usuario de tal aplicación y no por el resto de los integrantes del chat. De allí que Chatblocks encapsula esta responsabilidad en la contraparte local del ReceptionManager, el ReceptionView.
Los criterios de ordenación básicos que proporciona el framework son: por Autor, Por fecha, etc. Obviamente, el framework posibilita la extensión para definir los que se crean necesarios.

Floor Control
Un mecanismo de floor control [16], es decir especificar el protocolo de comunicación que se desea mantener entre los miembros: el modelo anárquico de los chat tradicionales no deja de ser en si mismo un protocolo, bastante relajado e irrestricto en casi todos los sentidos. Otro protocolo podría ser mas ordenado o restrictivo en cuanto al hecho de que usuario tiene permitido enviar mensajes, y en que momento. Este punto en particular es el que más va a diferenciar a una aplicación de otra, ya que determina el modo en que se llevará0 adelante la conversación

En Chatblocks tal componente encargado de controlar el curso de la conversación es el FloorControlManager.
Para ocupar este lugar, Chatblocks provee cuatro componentes concretos, cada uno de ellos tiene la finalidad de suministrar diferentes variaciones de este aspecto:
Sin Protocolo: Usando el componente *NoProtocolFloorControlManager*.
Con Protocolo: esta variante define un tipo de conversación en particular, como por ejemplo:

ProContraFloorControlManager para representar un debate sobre algún tema en particular entre dos personas, un Pro y un Contra, y varios oyentes, que conforman el publico.

AutoMediatorDiscussionFloorControlManager, para representar una discusión entre varias personas, por ejemplo una conferencia de prensa o una situación de participación de alumnos en una clase, la misma consta de participantes y un mediador, que es el encargado de dar la palabra, de a uno a la vez, a los participantes que la soliciten. En este caso el mediador es automático y posee una estructura FIFO para asignación de turnos.

Como una variante mas para este tipo de floor control contamos también con el *HumanMediatorDiscussionFloorControlManage*r, el cual en lugar de asignar los turnos con una política fija como la FIFO, se delega tal facultad a uno de los usuarios, el mediador.

Semántica de los mensajes
Para cubrir este aspecto, el framework provee la componente ShipmentManager, de la que existen tres variaciones, cada uno de estos componentes aporta una modalidad de este aspecto:

NoReferenceShipmentManager: Este componente cubre la modalidad de envío de mensajes sin semántica.

ThreadReferenceShipmentManager: Utilizando este componente como ShipmentManager se obtienen hilos o threads de conversación. Es decir mensajes que responden a otros.

ObjectReferenceShipmentManager: Este componente aporte al chat mensajes capaces de referenciar algún objeto, en este caso no existe restricciones de tipos de objetos.

El ShipmenteManager deberá tener en cuenta lo siguiente:

Formato de envío
El framework cubre una sola variación de este aspecto, ésta es texto plano, aunque puede extenderse fácilmente para soportar diferentes fuentes, colores y tamaños del texto a enviar.

Unidad de envío
En este caso, Chatblocks adopta como unidad de envío el *mensaje*. Esto es que los aportes de los participantes del chat se escriben en la máquina local y luego son enviados a los demás usuarios.

Por último, es imprescindible un administrador de los mensajes enviados, encargado de darle acceso al usuario a los mensajes ya enviados y nuevamente dependiendo de las posibilidades que desee brindar la aplicación final permita operaciones tales como la lectura, edición o eliminación de tales mensajes. Obviamente esta componente de la aplicación trabaja en estrecha relación con el repositorio de mensajes. El objeto que provee Chatblocks para esta funcionalidad es el ReceptionManager

Como toda aplicación COAST, además de las componentes compartidas, deben existir sus contrapartes locales, es decir, para aquellos aspectos que requieran de la participación del usuario es necesario definir una componente local que facilite la interfaz apropiada para su contraparte compartida.

Componentes Visuales
El framework provee dos mecanismos básicos de visualización de mensajes: en forma de lista(Acerca de cómo visualizar una como conversación puede consultar "Visualizing Conversation". [5]), la cual podría ser ordenada y filtrada de acuerdo a las necesidades de cada aplicación. Y en forma de árbol jerárquico, aplicando cierta relación entre el mensaje como por ejemplo, threads de mensajes (cada mensaje mantiene una referencia a su padre, puede interiorizarse mas en el tema leyendo Beyond Threaded Discourse [6]). En Chatblocks, esta componente es el ReceptionView.

Una interfaz que permita dar el awareness necesario sobre los usuarios que están utilizando la aplicación, es decir que brinde información acerca de quien es cada usuario, que rol cumple. Esta responsabilidad recae sobre el UsersActivityView.

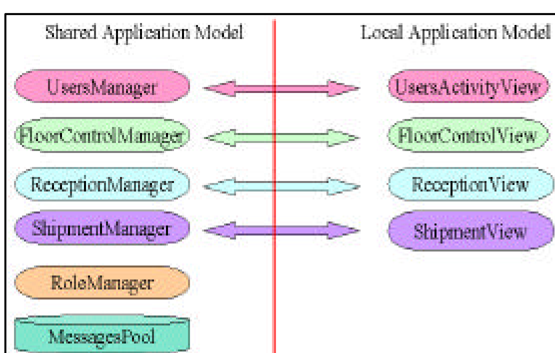
Una componente provista de los controles requeridos por el floor control para llevar a cabo el protocolo de comunicación que se haya dispuesto. Esta componente puede incluso carecer de cualquier tipo de control, para el caso de protocolos simples o libres que no imponen de ninguna restricción a los usuarios. Esta es la contraparte visual del FloorControlManager, el FloorControlView.

Otra componente encargada de permitir al usuario el envío de un nuevo mensaje. Además debe aportar un mecanismo de edición del mensaje. Esta componente será el ShipmentView.

El framework provee soporte para la creación de herramientas chat de forma rápida y precisa, ya que permite al usuario del framework, el programador de la aplicación, personalizar cada uno de estos aspectos.

3.1 Arquitectura General del Framework

Ahora que hemos detallado cada uno de los requerimientos involucrados en una aplicación del tipo chat, veamos en mas profundidad cada una de las componentes que forman parte del framework y como se combinan para dar como resultado una aplicación final.



Relación entre las componentes locales y compartidas

La idea de Chatblocks es aportar herramientas para la creación de aplicaciones chat, capaces de representar diferentes situaciones de diálogo. Esto se obtiene a partir de una estructura y distintos componentes.

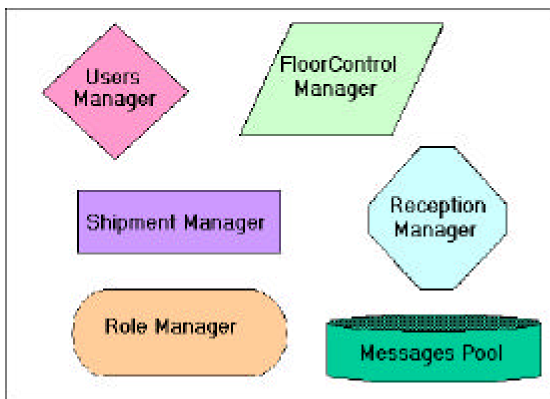
La estructura, forma el esqueleto de la aplicación, provee la funcionalidad común a todas las aplicaciones del dominio y en determinados lugares existen espacios en blanco. Cada espacio requiere de una funcionalidad especifica la que es ejecutada por un tipo de componente.

El framework provee distintas variantes para cada tipo de componente, formando una familia de componentes del mismo tipo.

Para realizar una aplicación que emule una conversación determinada basta solo con incluir los componentes que satisfagan las situaciones requeridas por la aplicación. Por ejemplo, si necesitamos realizar un chat que emule un debate de TV, deberíamos incluir un componente que otorgue turnos alternados, pero si en cambio queremos emular una discusión de una conferencia de prensa, deberíamos incluir un componente que seleccione entre los participantes con el pedido de palabra y se la otorgue a uno de ellos.

El framework se divide en dos capas una local llamada (LocalApplicationModel) que se compone de todas las partes locales de la aplicación, tales como interfaces y eventos. Y una parte compartida (SharedApplicationModel) en donde se encuentran las componentes a compartir por todos los usuarios.
En la capa compartida se encuentran los managers, cada manager cubre uno o más aspectos de la aplicación final, los managers interactúan entre sí, para llevar a cabo las operaciones que se les encomiendan. Estos realizan operaciones directamente sobre el MessagesPool, que es el repositorio de mensajes, agregando o extrayendo de este, los mensajes que necesiten para completar las operaciones.
Las views poseen una comunicación directa con los managers, reflejando los cambios que sufre el corazón de la aplicación, por ejemplo, nuevos mensajes, ingreso y egreso de participantes, cambios de turnos, etc. Cada view interactúa con un manager específico, de éste recibe la información que debe mostrar al usuario y hacia éste envía las operaciones que el usuario ordena a la aplicación.
La interfaz del usuario surge de la composición de views, mientras que el corazón de la aplicación, la parte compartida, surge de la composición de managers y el MessagesPool. Para más información acerca de internases alternativas puede consultar "Alternative Interfaces for Chat" [3]

3.1.1. Componentes Compartidos



Administradores compartidos

MessagesPool
Es el repositorio de mensajes. Su función es mantener todos los mensajes que se intercambian a lo largo de la conversación.
Cuando un mensaje es enviado el ShipmentManager lo agrega al MessagesPool, el ReceptionManager retira todos los mensajes de este repositorio cuando debe entregar los mensajes al usuario. Consta de una colección, donde se almacenan los mensajes, llamada messages, y las operaciones necesarias para la manipulación de los mismos, es decir, agregar eliminar y buscar mensajes entre otras.

Message
Representa un mensaje. Posee tres atributos principales, Autor(Referencia a un ChatUser, que es el emisor del mensaje),Text (Es un String, el mensaje en sí) y CreationTimestamp (Fecha y hora de creación.). Si se quiere, se puede extender message incorporándole otros atributos, como por ejemplo "parent", para brindar la posibilidad de crear mensajes con referencias a los padres (por ej.: ThreadReferenceMessage) de esta forma se tienen cadenas de mensajes.

ChatUser
Representa al usuario del chat. Posee una imagen (icon), la cual se puede utilizar como avatar y una instancia de UserRole, que representa el rol del usuario en el chat.
La forma de asignación de los roles puede ser aleatoria o predefinida, teniendo para esto una colección de roles y usuarios a los que se les deben asignar esos roles.

UserRole
Representa el rol de un participante del chat (pro, contra, oyente, mediador, etc), dependiendo del tipo de conversación que queremos representar (debates, charlas, etc), debemos modelar diferentes familias de roles de usuarios. El framework provee algunos tipos, que se detallan a continuación, en

caso de querer agregar una familia de roles o un rol simple, solo basta con agregar una subclase concreta de UserRole y una subclase concreta de RoleManager que asigne estos nuevos roles de usuario. Se pueden representar diferentes roles tales como DiscussionUserRole (modela los roles que existen en una discusion), ProContraUserRole (modela los roles que hay en un debate en donde un participante es pro y otro es contra), etc.

ChatSharedApplication
Conforma la parte compartida del chat. Sirve de esqueleto para las futuras aplicaciones. Los atributos de ésta clase referencian a solo una de las componentes del chat. Para personalizar estas referencias solo basta con crear una subclase de ChatSharedApplication y rescribir los métodos de iniciación de los slots a personalizar.
El framework provee, en algunos casos varios componentes para cumplir una determinada función:

UsersManager
Mantiene información sobre los participantes del chat, se encarga de asignar los avatars de los usuarios a los mensajes.
Mantiene una colección con todos los participantes de la conversación (ChatUsers) y otra con los participantes que se encuentran conectados (OnLine).

RoleManager
El RoleManager es el encargado de asignar los roles a los usuarios, éstos son subclases de la clase UserRole, que es una clase abstracta.
Las subclases de UserRole forman familias de roles de usuario. Para cada familia de roles existirá un RoleManager especifico, subclase de RoleManager, por ejemplo, el DiscussionRoleManager asigna roles del tipo DiscussionUserRole.
La forma de asignación de los roles puede ser aleatoria o predefinida, teniendo para esto un diccionario con roles y usuarios a los que se les deben asignar esos roles.
El framework provee tres componentes RoleManager (DiscussionRoleManager, NoRoleRoleManager y ProContraRoleManager) ya predefinidos, pero se pueden agregar mas a voluntad.

FloorControlManager
El FloorControlManager es el encargado de mantener el control de la conversación en la aplicación, y determinar que usuarios están habilitados para realizar operaciones. Cada protocolo que se desee modelar implica un floor control diferente.
En el caso de necesitarse un protocolo diferente a los que provee Chatblocks, este puede agregarse al framework. Para esto se deberá crear una subclase de FloorControlManager. Por lo general, cada protocolo necesita una familia de roles de usuario, por eso para cada implementación de protocolo, se deberá implementar una familia de roles y su respectivo RoleManager.
Al FloorControlManager le llegan los evento de entrada y salida de los usuarios al chat (estos los envía el UsersManager) y envíos de los usuario (capturado por el ShipmentManager). Para agregar la captura de otros eventos, se debe modifica el manager correspondiente, que sería el encargado avisarle al FloorControlManager del evento ocurrido.

Shipment Manager
Este componente es el encargado de realizar las operaciones de envío de los aportes de los usuario, por medio del método **send.** La operación consiste en tomar los mensajes, que son solo strings, y crear un objeto del tipo **Message**, y luego agregarlos al MessagesPool, de donde luego se tomarán los mensajes para su lectura.

En cuanto al mensaje, éste puede ser una instancia de la clase **Message** o alguna subclase de la misma, todos los componentes concretos responden al send, creando mensajes de la clase Message, luego cada uno agrega operaciones para agregar semántica a los mensajes.
El framework define por defecto los siguientes ShipmentManagers:

NoReferenceShipmentManager: Envía mensajes simples, sin referencia, esto implica que los mensajes no poseen semántica. Sólo responde al send de la superclase.

ThreadReferenceShipmentManager: Envía mensajes que referencian a un "padre" o "antecesor" en un thread o hilo de conversación, los threads se van armando con las referencias de mensajes a otros. Estos forman una estructura jerárquica, ya que varios mensajes pueden referirse a un mismo mensaje. Los mensajes sin referencia a un padre crean un thread, esto quiere decir que introducen un subtema a la conversación, en cambio los que referencian a un padre están continuando un thread.

ReceptionManager
Es una clase concreta. Este componente es el único que provee el framework para cumplir esta función. Es responsable de la recepción de mensajes y del filtrado de los mismos. La recepción de los mensajes consiste en tomar los mensajes del MessagesPool, estos pueden ser filtrados o no.
Para el filtrado de los mensajes existe un subcomponente, lamado **MessageFilter** cuyo objetivo es reconocer, en una colección de mensajes dada, cuáles son los que cumplen con determinado criterio. Cada usuario puede poseer un objeto MessageFilter diferente como atributo, asi cada uno puede realizar filtros por diferentes criterios.

3.1.2. Componentes Locales: Los View Managers

ChatLocalApplication
Es la contraparte local de ChatSharedApplication, esta forma la interfaz de usuario. Esta es una clase abstracta, y posee cuatro atributos, que son slots Coast, cada uno de estos referencia a un componente de la interfaz. Los componentes locales son elementos gráficos que al combinarlos forman la interfaz de usuario, cada tipo de componente tiene su lugar asignado en la interfaz. El framework provee, en algunos casos, varios componentes para una determinada ubicación.
Estos componentes reflejan los cambios en el DomainModel, cuando se enteran de un cambio le piden al componente correspondiente del SharedApplicationModel que les retorne la información para luego refrescar la interfaz de usuario.

User ActivityView
Exhibe la información de awareness que manipula el UsersManager: Esta componente consta de una lista donde muestra todos los usuarios que participan de la conversación, donde los usuarios conectados (On-Line) se muestran con el estilo de fuente bold y los desconectados con estilo normal.
Para realizar esta función posee dos slots computados, uno es la colección de participantes y el otro consiste en la colección de usuarios On-Line.
Este componente es el único que provee el framework para exhibir esta información, como idea a realizar en una futura extensión del framework es agregar un componente que muestre los iconos de cada usuario, en lugar de sus nombres.

ReceptionView
Es responsable de mostrar los mensajes que recibe un usuario.
A su vez esta compuesto por dos subcomponentes viewer y ordered. Estos son instancias de las clases **MessageViewer** y **MessageOrderer** respectivamente. El primero aporta la representación gráfica, mientras que el segundo agrega lógica para la ordenación de los mensajes.

Los mensajes son solicitados al ReceptionManager, este los entrega en una colección, la misma puede estar filtrada o no, esto depende del método que se utilice para solicitar los mensajes. Una vez obtenidos los mensajes, la colección que los contiene pasa por el "orderer", que ordena la colección dependiendo por un criterio determinado, dependiendo de la clase a la que pertenezca la instancia del mismo.
Luego del ordenamiento de la colección de mensajes esta pasa al viewer, donde se muestran al usuario, la forma de visualización será en forma de árbol o lista, dependiendo de la clase instanciada para realizar este trabajo.
Como viewer es posible utilizar cualquiera de las subclases de **MessageViewer,** estas son:
MessageViewerList, muestra los mensajes en forma de lista. Y **MessageViewerTree,** muestra los mensajes en forma de árbol. Los mensajes que posean algún atributo en común, dependiendo del criterio de ordenación utilizado, formarán una rama del árbol.
Los viewers se pueden combinar con los orderers, obteniendo como resultado nuevas formas de visualización de mensajes. Por ejemplo se podría querer ordenar por Fecha, Autor, Thread, etc.

ShipmentView
Este componente es el encargado de despachar los mensajes al **ShipmenteManager** para que este último complete la operación.
La interfaz de usuario consta de un campo de texto, donde se escribe el mensaje a enviar, y un botón, que es el que realiza la operación de envío.
El envío esta orientado a mensajes, ya que el usuario primero lo escribe en el campo de texto y luego lo envía presionando el botón send, o la tecla enter.
El formato de los mensajes es texto plano, ya que esta versión del framework no provee asignación de estilos y tipos de fuentes a los mensajes.
Este componente posee lógica para habilitar y deshabilitar el botón send, con la finalidad de indicar que el usuario puede enviar mensajes o no. Estas funciones están implementadas para brindarle apoyo al FloorControlView.
Esta una clase abstracta, y de ella derivan tres subclases concretas:
ThreadReferenceShipmentView: Al enviar un mensaje interactúa con el ReceptionView, y si existe algún mensaje seleccionado, el nuevo mensaje se transforma un la respuesta al mensaje seleccionado. Si no existe selección el mensaje se convierte en un nuevo thread, no posee referencias a mensajes previos.
NoReferenceShipmentView: Envía mensajes simples, sin referencia.
ObjectReferenceShipmentView: Envía mensajes que referencian algún objeto, no existiendo restricciones sobre la clase de este objeto.
La interfaz de estos tres componentes es la misma, ya que todos la heredan de la superclase abstracta, las subclases no agregan componentes visuales, solo agregan lógica.

Cada uno de estos componentes concretos fueron diseñados para poder interactuar con cada uno de los componentes ShipmentManager concretos, esto es, si se usa un determinado shipmentManager se debe utilizar un determinado ShipmentView:

FloorControlView
La principal función de este componente es informar sobre el desarrollo del protocolo de la aplicación, esto es, si este existe, mostrar el estado en que se encuentra el usuario actual. Si se desea, se puede usar o crear componentes concretos de esta view para interactuar con algún floorControl en particular. El Floor Control por defecto es el NoFloorControl, el cual especifica un protocolo anárquico equivalente a los chats tradicionales. Un ejemplo de un FloorControl podría ser

el del ProContraFloorControlView. Este componente muestra tres etiquetas, las cuales se detallan a continuación:
En el sector derecho muestra cual de los participantes del chat tiene el rol de Pro y cual el de Contra. En el sector izquierdo se muestra que rol posee el usuario local.
A medida que transcurre la conversación, el envío de mensajes al usuario le es permitido a determinados usuarios, como ya se dijo cuando se explico este protocolo, para proporcionar feedback al usuario este componente interactúa con el ShipmentView habilitando y deshabilitando el botón de envío, indicando si el usuario puede o no realizar esta operación.
Por ser un framework de caja blanca, la extensión se realiza por medio de sub clasificación de clases existentes, permitiendo al usuario del framework extender o agregar componentes.
Cada uno de estos componentes esta diseñado para actuar como contrapartida local de un determinado FloorControlManager.

4. Conclusiones

Chatblocks ha demostrado ser capaz de crear herramientas chat innovadoras. Por basarse en el framework de groupware COAST independiza al diseñador de chats del modelo de comunicación subyacente. La flexibilidad obtenida y la capacidad para soportar protocolos explícitos, en las aplicaciones finales, serán de gran utilidad a la hora de experimentar con nuevas herramientas.

Chatblocks es actualmente utilizado por el instituto Fraunhofer en Alemania par la construcción de variedad de aplicaciones chat. A continuación describimos las principales consecuencias del uso del framework.

Facilidad: Con Chatblocks el programador de la aplicación no debe conocer internamente la estructura del framework, solo conocer la funcionalidad de cada uno de los componentes provistos por el mismo. Con lo cual el costo de aprendizaje es bajo, ya que si la arquitectura es lo suficientemente fácil de entender, los tiempos de aprendizaje serán más cortos lo cual lleva a un bajo costo económico para el desarrollo.

Rapidez: La construcción de una herramienta chat consiste en seleccionar que componentes se desean usar, o sea crear instancias de los componentes y agregarlos al chat, como lo hace un programador de GUIs cuando, selecciona que widgets necesita utilizar, crea sus instancias y las agrega a su GUI. Permite una rápida recuperación del tiempo que se necesita perder para el aprendizaje de la arquitectura. La idea es que al momento de crear un chat, el programador decida solo que componentes se desean usar.

Flexibilidad: Aporta mecanismos para la experimentación y evaluación de variaciones de herramientas chat. Ejemplo, protocolos explícitos, roles de usuario, formas de visualización, representaciones de usuarios, etc. Todos estos aspectos se pueden combinar, obteniendo un gran numero de chats posibles, sin necesidad de extender el framework. Por ser un framework, chatblocks permitirá futuras extensiones, aumentando el nivel de flexibilidad a la hora de crear una nueva herramienta chat.
Extensionalidad: El framework permite la creación de nuevos componentes, por ser un framework de caja blanca, esto se obtiene mediante la subclasificación de los componentes provistos por Chatblocks. Logrando así poder modelar mas protocolos, roles, diferentes interfaces de usuario, etc.

Soporte para la implementación de protocolos explícitos: Uno de los atractivos más importantes, de las herramientas chat que se podrán crear con Chatblocks, es el hecho de poder proporcionar protocolos explícitos, para ello se necesita soportar una serie de aspectos y re-adaptarlos según la

necesidad. El prototipo ha comprobado que Chatblocks permite implementar un protocolo explicito, el framework provee dos protocolos Discussion y ProContra.

Usos: Entre los usos para los que se utilizará este framework podemos citar:
Evaluación: Se puede utilizar para evaluar la comunicación y aspectos del comportamiento entre los individuos.
Educación: Se puede utilizar para generar herramientas para el aprendizaje. por ej.: aprender nuevos idiomas .

5. Trabajos futuros

La primera versión del framework se realizo en Smalltalk, y como ya dijimos basado en COAST, lo próximo es construir una versión estable del framework en Java, el mismo estará realizado sobre un framework cooperativo con características similares al COAST, llamado DyCE. Permitir el envío de mensajes con tipo, enriqueciendo la semántica de los mensajes. Mejorar la interfaz de usuario mediante la realización de widgets específicos para la versión Java. Extensión del modelo para soportar estadísticas sobre los mensajes enviados y recibidos por los usuarios. Agregar mecanismos de monitoreo de actividad del usuario (mas awarenes), para detectar situaciones en las que la aplicación se encuentra activa pero el usuario no se encuentra observándola. Ej. detectar si el usuario esta en estado "Away". Otra de las futuras modificaciones que se desean realizar como trabajo futuro es transformar a Chatblocks en un framework de caja negra.

6. Referencias

1. M. Fayad, D. Schmidt. "Object Oriented Application Frameworks". Communications of the ACM Octubre 1997/Vol. 40, No 10, page 32.
2. R. Johnson. Frameworks = Componets + Patterns. Communications of the ACM Octubre 1997/Vol. 40, No 10, page 39.
3. D. Vronay, M. Smith, S. Drucker. Alternative Interfaces for Chat. CHI'99.
4. F. Viegas, J. Donath. Chat Circles. CHI'99.
5. F. Viegas, J. Donath, K. Karahalios. "Visualizing Conversation". MIT Media Lab. 1999.
6. J. Hewitt. Beyond Threaded Discourse. WebNet'97.
7. Douglas C. Schmidt, Mohamed E. Fayad, Ralf E. Johnson. Building Application Frameworks.
8. Fayad, M.E., Schmidt, D.C., and Johnson, R.E. Object-Oriented Application Frameworks: Problems and Perspectives. Wiley, NY, 1997.
9. Ellis, C., Gibbs, S., and Rein, G., Groupware: Some Issues and Experiences, Communications of the ACM, Vol. 34, No. 1, 1991, 39-58
10. Ackerman, M., Starr, B., Social Activity Indicators for Groupware, Computer, June 1996, 37-42
11. Altun, A., Interaction Management Strategies on IRC and Virtual Chat Rooms, SITE 98 proceedings, March 10-14, 1998, 1223-1227
12. Beaudouin-Lafon, M., Karsenty, A., Transparency and Awareness in Real-Time Groupware Systems, UIST proceedings, 1992, 171-180
13. Dourish, P., Bellotti, V., Awareness and Coordination in Shared Workspaces, CSCW proceedings, 1992, 107-114
14. S., Portholes: Supporting Awareness in a Distributed Work Group, CHI proceedings, 1992
15. COAST: http://www.opencoast.org
16. Usability first: http://www.usabilityfirst.com/glossary