A TOOL FOR ROLE-BASED CHATTING

Haibin Zhu, Senior Member, IEEE and Rob Alkins
Department of Computer Science and Mathematics, Nipissing University,
100 College Dr., North Bay, ON, P1B 8L7, Canada
haibinz@nipissingu.ca, alkins.rob@gmail.com

Abstract

Role-based collaboration (RBC) is proposed to support people’s collaboration with more usable human-computer interfaces. To approach the objectives of RBC, new practical tools are required. Designing, implementing and using the new tools are the major methods to accomplish the tasks of research on RBC.

This paper presents a tool for role-based chatting that is a typical instance of role-based collaboration. The scenario of role-based chatting, and the tool’s architecture and implementation are described. The chatting tool reflects all the principles outlined by role-based collaboration. This tool shows that role-based collaboration is practical, and feasible. It also shows the possibility of building more complex role-based systems.

Keywords: Roles, Role-based, Chatting, Tool, Role-Based collaboration.

1 Introduction

Computers are everywhere and computer-based collaboration is becoming a basic skill of people. Now, most people complete their daily routine tasks with computers. Emailing and surfing on the Internet are now common daily activities. However, there are still many people who do not like using computers to work because they think computers are awful and they never use computers or computer-based tools if they do not have to. This requires computer scientists and engineers to create more usable software and hardware products to help and encourage people to work with computers. If it is much better for people to work with a computer-based tool than not with it, they definitely like to use the tool.

Role-Based Collaboration (RBC) [11] is a methodology to design and implement new computer-based tools. It is an approach that can be used to integrate the theory of roles into Computer-Supported Cooperative Work (CSCW) systems and other computer-based systems. It consists of a set of concepts, principles, mechanisms and methods. RBC imposes challenges and benefits not found in traditional CSCW systems. Role-based collaboration can help people in both long-term and short-term collaboration. It will bring in the benefits as follows:

In long-term collaboration, roles help
- Identify the human user “self” [3];
- Avoid interruption and conflicts [5, 8];
- Enforce independency by hiding people under roles [1];
- Encourage people to contribute more [6, 9]; and
- Remove ambiguities to overcome expectation conflicts [2, 4, 11].

In short-term collaboration, roles help
- Work with personalized user interfaces [7];
- Concentrate on a job and decrease possibilities of conflicts for shared resources [11];
- Improve people’s satisfactions with more peoples’ playing the same role during a period; and
- Transfer roles with the requirement of a group [12].

In management and administration, roles help
- Decrease the workload of system administrators;
- Separate of concerns in designing;
- Decrease the knowledge space of searching;
- Create dynamics for components; and
- Regulate ways of collaboration among agents.

From the above benefits, RBC research will bring exciting improvements to the development and the application of CSCW systems, and the methodologies of collaboration. There are requirements for tools to facilitate roles and interaction among roles.

This paper is arranged as follows: Section 2 discusses the benefits of role-based chatting; Section 3 describes the architecture of the tool; Section 4 presents the implementation of the tool, Section 5 concludes the paper and proposes the topics for future research.

2 Role-Based Chatting

Collaboration emerges in different forms. It is no doubt that chatting is a common form of collaboration. Without the help of computer-based tools, it would be impossible to practice role-based chatting. In role-based chatting, every person plays one or more roles and the people who are in chatting do not have to know each other (Fig. 1).

In reality, client service is widely used in technical services of a company. Such a service is generally facilitated by phones or emails. For clients, this service is role-based, because they do not care who answer their questions, even though the person who answers the technical questions begins by telling the client his/her name, in fact, first name. What the clients only concern is the qualifications of the service providers. There might be such an argument that telling names can activate the service providers to work better. In fact, by role-based chatting, the service providers have the same driving force to serve, because it is easy for the managers to recognize who provides a specific service by role scheduling.
In role-based chatting, it is also easier for a manager to find a technician then to find a specific person to serve. On the other hand, using role-based chatting, two or more technicians can chat with one client with the same role to improve the efficiency of a specific service and make the client feel more comfortable and more satisfactory. Further, role-based chatting can encourage people's participation by specially designing some facilities such as credits [6, 9].

Chatting can encourage people's participation by specially designing some facilities such as credits [6, 9]. Further more, role-based chatting can chat with one client with the same role to improve the other hand, using role-based chatting, two or more technicians a technician then to find a specific person to serve. On the other hand, using role-based chatting, two or more technicians can chat with one client with the same role to improve the other hand, using role-based chatting, two or more technicians can chat with one client with the same role to improve the.

Further more, role-based chatting can encourage people's participation by specially designing some facilities such as credits [6, 9]. Further more, role-based chatting can chat with one client with the same role to improve the other hand, using role-based chatting, two or more technicians a technician then to find a specific person to serve. On the other hand, using role-based chatting, two or more technicians can chat with one client with the same role to improve the other hand, using role-based chatting, two or more technicians can chat with one client with the same role to improve the other hand, using role-based chatting, two or more technicians can chat with one client with the same role to improve the other hand, using role-based chatting, two or more technicians can chat with one client with the same role to improve the.

In role-based chatting, it is also easier for a manager to find a technician then to find a specific person to serve. On the other hand, using role-based chatting, two or more technicians can chat with one client with the same role to improve the efficiency of a specific service and make the client feel more comfortable and more satisfactory. Further more, role-based chatting can encourage people's participation by specially designing some facilities such as credits [6, 9].

This kind of chatting tool actually forms a positive community that encourages people to help each other without knowing each other. Eventually, although they never know each other, they get a fair evaluation by the system. The people in the chatting are satisfactory and feel comfortable in participating the chatting.

7) The people in chatting are evaluated by their credits collected by playing roles in the chatting. This tool presented is actually a simplified instance of our E-CARGO model [11] where roles represent a title with both responsibilities and permissions, and agents represent human users of the tool. A role is defined as $r ::= <n, M_c, M_r>$ where, $n$ is the identification or the name of the role, $M_c$ denote sets of services or responsibilities, and $M_r$ the requests or rights [10]. Here, $M_c$ and $M_r$ specify the extent of the messages issued or received by a user. A message is defined as $m ::= <n, v, d, a, f, t>$ where $n$ is the identification of the message, $v$ is the sender role, $d$ and $d_r$ express the sender agent and senders’ role, $f$ is the message text by the user, and $t$ is a tag that expresses any, some or all message [11]. An agent is defined as $a ::= <n, s, w, N_r>$, where $n$ is the internal identifier for the agent, $s$ expresses the profile of the relevant user, $w$ the credit, and $N_r$ is the list of roles assigned to this agent [9].

It supports long term anonymous collaboration in a dynamic, event-driven environment. Features include the encouragement of participants’ participation with credits for contribution [6, 9], denial of payment for insufficient contribution and an appeal mechanism to prevent cheating. The emphasis is on two collaboration mechanisms: messages and dialogs. A message is a one-shot collaboration that involves only two agents with a request and a reply. A dialog is an ongoing collaboration that can include any number of agents.

3 The Architecture of the Role-Based Chatting Tool

As what discussed in [10, 11], a role-based collaborative system should be built on client/server architecture. In this tool, all the foundation concepts discussed in Section 2, such as, role, agent, message, permissions and responsibilities, are designed as classes and the design of server and client is based on these foundation classes. The tool is composed of a server, many clients and an SQL server (Fig. 2).

The server provides the management of information and controls the interactions among client users. It sends and receives information and requests, and makes calls to SQL
server in order to manage all the information of the system including roles and agents (Fig. 3).

Based on the client/server architecture, all the information in the system is stored in an SQL server database with associated date and time entries. In this way, at any given moment, the state of the system is completely specified by the information on the SQL server which is installed in a powerful server computer. Using this approach, there is no need to save the system information on the server upon its shutdown, nor is there any critical data that may be lost on the server due to a power failure. All the updates to the system are preformed in a single database command. For example, when a user sends a message, the message is first stored in a database table entitled “post office”, it remains there until the MessageDispatcher thread obtains appropriate recipients, only then is the data moved to the recipients’ mailboxes. In addition to the benefit of the increased stability, the server only requires a small amount of volatile memory (an integral commodity on large server systems).

The client side (Fig. 4) has a main class Client. This class creates a WorkspaceWindow at the beginning when a user logs in. In the WorkspaceWindow, a RoleList and a RoleEditor can be opened for chatting with a specific role and the AgentList and the AgentEditor are opened for special roles to manage. The class ClientConnection provides the connection between the server and the client. The WorkspaceWindow contains a MessageList that includes many MessageWindows. Any changes to the state of the system are also updated immediately on the client systems in real time such that it is not necessary for the user to reload any interface windows. This means that the server is not completely passive but actively sending updates to the clients.

4 Implementation

4.1 User Interfaces

The user interfaces are in window-style. A new user can be added to the system by creating an agent. In the following discussion, user and agent may be used interchangeably. Normally, user is used to express the real person and agent is used to express the system entity relevant to a user. A user’s permissions are a union of the permissions associated with each role the user is playing. The functionality of the interface presented to the user is governed by the permissions associated with each role the user is playing. The main user interface is a small window that contains the users’ current credits and buttons to access features such as a role list, mailbox, message window, and dialog list. The role list allows users to view roles. The mailbox is where any incoming messages are displayed. The dialog list displays any dialogs that the user is participating in. When a dialog is selected, a list of the participants in that dialog is presented. If the user is the creator of the dialog, s/he has access to the buttons to add or remove roles to the dialog. Each of these interfaces is dynamically updateable, and any changes to the users’ permissions, credits, or inbox are immediately updated on open windows in time (Fig. 5).
When a user composes a new message, a drop down list of the roles s/he is playing is presented. The user chooses the role s/he wishes to play when sending the message. A drop down list of all roles on the system allows the user to select the desired receiving role. The input fields are presented for the number of credits that the sender wishes to award the receiver if an appropriate reply is received, and the number of desired recipients of the message. The user may enter the message text at any time but the message may not be sent until all the required information has been entered.

When the recipient of a message opens a message from their mailbox, all of the information that the sender supplies is displayed but un-modifiable, only a text area for the recipient’s reply is available for modification. The message is then marked as a reply and sent back to the sender. Upon receiving a reply, only two options are presented: to either pay, or to deny payment. If “Pay Agent” is selected, a number of credits corresponding to the value of the message are moved from the sender’s credits to the receiver’s credits.

If the sender of a message receives a reply that the sender deems insufficient, the sender may choose to deny payment, in which case the message is marked “denial of payment” and sent back to the user who initially received the message and composed the reply. The recipient is then presented with two options: to accept the sender’s denial of payment, or to appeal the decision of the sender. Appeals are a separate kind of message whose recipient role can be chosen only from those that have the “Appeals Manager” obligation.

When a user creates a dialog (Fig. 6), s/he must choose the role that s/he wishes to play within this dialog and assigns a subject to that dialog. Only then can a user compose “Dialog Invitation” messages. Dialog invitations, composed only by the creator of a dialog, consist of the desired recipient role, the number of agents playing those roles whom should receive invitations and the value of the message. A text field is supplied for the dialog creator to outline what is expected of the participant if they are to receive the payment that is specified by the invitation message. Once an agent joins a dialog, they must either be sent a denial of payment or paid the value of their invitation message before they are removed from the dialog or the dialog is closed. The denial of payments is, of course, open to appeal.

The dialog window itself is also dynamically updateable so collaboration may occur in both real time and long term. All entries in the dialog are displayed with the sender role displayed in bold followed by the body of the dialog entry. It is possible for two agents to play the same role within the same dialog, which has been argued as a benefit in Section 2. However, an option is provided to distinguish different agents with the same role by a number assigned when they join the dialog displayed in parenthesis. Any entries made by the user who is viewing the dialog are displayed in red and have the word ‘Me’ displayed in parenthesis. System messages such as an agent joining a dialog are displayed in blue.

### 4.2 Client Interactions

Both the client and server programs are written in Java using the Standard Widget Toolkit (SWT) widgets.

For each client connection, there is a dedicated thread on both the server and the client. Both threads have an outbox queue where network messages from other threads are stored until the communications thread is done dealing with any incoming network messages. This approach avoids any concurrency issues with different threads attempting to communicate simultaneously. Any network messages that require a reply are numbered with a distinct long integer such that message replies can be paired with the appropriate request.

Network messages are passed as strings that are terminated by an escape character, thus message strings may include a new line. Each network message begins with a message header that determines the context of the rest of the message. For example, a login message from the client begins with “LOGIN” header that is immediately followed by a username, password and request number. The client thread that initiates the login then waits for a reply with the same request number. If the login is successful a “LOGIN_CONFIRMED” message consisting of the request number, an agent object and a permissions object is sent from the server. The login message strings may include a new line. Each network message begins with a message header that determines the context of the rest of the message. For example, a login message from the client begins with “LOGIN” header that is immediately followed by a username, password and request number. The client thread that initiates the login then waits for a reply with the same request number. If the login is successful a “LOGIN_CONFIRMED” message consisting of the request number, an agent object and a permissions object is sent from the server. The login initiating thread is then notified that it may resume execution.

Classes such as Agent, Dialog, Message, Permissions, Responsibilities, Role and ListResource (general purpose collection of strings) implement an interface called Resource.

```java
public interface Resource {
    public void receive(NetworkReader in);
    public String[] toStrings();
    public String getType();
}
```

This requires all objects of the above classes to implement the receive(...) method in the manner such objects may be passed freely over the network. Some of the above classes may contain other classes of objects, i.e., a role instance contains a Permissions instance and a Responsibilities instance.
The functionality of the system is primarily driven by the client application, objects are maintained on the server by means of REQUEST_RESOURCE, APPEND_RESOURCE and DELETE_RESOURCE network messages that correspond to request, delete, and append properties of the Permissions object. Although this method of access control was intended to be the only access control during early development it was noted that there was a need for many special cases that have to be treated in a less general manner. The above messages apply to objects that are global to the system i.e. the collection of roles and agents or dialog headers.

A dialog object in the system consists of: an identification string, entry number, an agent string, a role string, a date and time string, and the dialog text string. In an attempt to treat dialogs in a general manner, a dialog with an entry number 0 is treated as a dialog header. As more functionality was built into the dialog mechanisms, more data was needed in the dialog header. This caused a conflict of interest between adding more data fields (to both the dialog objects and the database representation of them) and the wasted space that would be contained in dialog objects that were not a header. As the system matured, it became clear that separate object types should have been created. Currently, a dialog header has: the dialog subject in the role field, the agent field contains the dialog creator and the data field contains a list of agents and the role they are playing in that dialog on alternating lines.

The real time updating of the client window is accomplished by a network message that is initiated at server side: the “RESOURCE_CHANGE” header is followed by the type of resource that has changed and the index of the resource. Any client window that has dynamically updateable content implements an interface called updateable, requiring the class to implement an update method. It is through this interface that the client connection thread may execute synchronized code within the Graphical User Interface (GUI) thread by means of the SWT Shell.asyncExec(...) method.

4.3 Server Management

To facilitate role-based chatting, management is an important job. Agent management manages all the uses profiles and relevant roles the relevant users are currently playing and have played in the past. Role management deals with all the roles in the system.

The main administration interface is a window that is same as a user interface but with more features such as role and agent management. The role list allows administrators to manage roles. The agent list allows the same actions to be performed on agents. The basic administrative operations for role management are as follows (Fig. 7):

- Add/view/edit/delete a role or an agent;
- Approve/reject an application for role;
- Assign /delete a role to/from an agent;
- Adjust the credits of a person; and
- Dispatch messages to people.

A new agent is created with a unique username, password, number of credits, and an empty list of roles. There are default roles for a new agent. Roles can then be added to the agent. With each role that is added to an agent, the user’s permissions and obligations grow accordingly.

Role and agent management is implemented through client connections. If a particular user has a role with associated permissions such as add/delete agents, his/her agent may make changes to the agent table in the database. This allows for a connected client to change an agent’s password, credits or add/remove roles to that agent. Likewise, if a user has a role that possesses the permission to append/delete roles, his/her agent may change the permissions or obligations for that role.

Fig. 7 The Role Management Interface
To provide a server for chatting, all the information relevant to users and administrators are stored in the SQL server at the server side. Therefore, all the above management operations will directed to a management class at the server side, i.e., the SQLInterface class that performs the operations transferred from the Server class to the SQL server database (Fig. 3). Each request to read or update the database is performed in single request. In this manner concurrency issues are mainly handled by the SQL server database.

Multithreading is present more in the server application than the client. The classes MessageDispatcher, ConnectionHandler and ServerConnection all extend Thread and implement the necessary run() method. Upon startup, the server instantiates a ConnectionHandler thread that listens on the designated port for any incoming connections. Any incoming connections cause the ConnectionHandler to instantiate a ServerConnection thread, this thread provides the service interface for any connected client threads.

Prior to a client’s logging in, the permissions are set to the default (minimal) values. Once a login message from the client is received, the server will query the database for any agent instances with a username and password matching those contained in the message from the client. A successful search will return an agent with a list of roles, each role instance is loaded from the database and the permissions of the newly connected agent are collected by taking the union of all the permissions associated with each role in the agent instance. This permission objects and the agent object are then transmitted to the client.

The MessageDispatcher thread continuously queries the database for any messages in the “Post Office” table. Any newly found messages are placed in the mailbox of an appropriate recipient (agent). Appropriate recipients are chosen first from the connected agents, then, if none is available the non-connected agents are checked, any non-deliverable messages remain in the “Post Office”. Any transfers of a message to a client mailbox are collected in a single database command, avoiding any loss of information from concurrency problems or software failure.

5 Conclusions and Future Work

We present the design and implementation of a role-based chatting tool. As this is a prototype, there is some work left to be investigated. The security of the system is not the primary consideration during the current design and implementation. The client does not perform any action that they do not have the rights to perform. Appropriate permissions, violations and messages are sent from the server to the client. However, messages or dialog entries passed to the client contain the username of the sender in order to facilitate ongoing communication. This information is sensitive if anonymity of the system is to be preserved. The client does not relay this information to the user but it is transferred to a potentially unsecured client.

We also need more comprehensive investigation for the usability and affordance of this tool to evaluate its real value in supporting people’s chatting. Before this investigation, we should first make the prototype an online tool that can be easily accessed from the Internet and used in a 365/24/7 style to support collaboration anywhere and anytime.

Acknowledgements

This research is supported by the IBM Eclipse Innovation Grant Funding and NSERC (National Science and Engineering Research Council of Canada, No. 262075-06). Thanks also go to Daniel Plourde who participated in the early stage of this work.

References