The Transparent Adaptation Approach to the Development of Awareness Mechanisms for Groupware

Minh Hong Tran, Yun Yang
FICT, Swinburne University of Technology
{mtran, yyang}@ict.swin.edu.au

Gitesh K. Raikundalia
ITArl, Victoria University
Gitesh.Raikundalia@vu.edu.au

Abstract

Implementing support for group awareness is an essential and challenging process in groupware development. This paper reports our research on developing a Transparent Adaptation *(TA) approach, which is used to develop awareness mechanisms for groupware. The TA approach is distinctive because the implementation of awareness mechanisms is separate from that of groupware (*groupware transparency*), and components developed in this approach can be reused in building different awareness mechanisms (*software reuse*). In this paper, we describe our software architecture for the TA approach, and four software layers of awareness mechanisms. We have applied the TA approach in implementing awareness mechanisms for collaborative word processing (CoWord) and instant messaging (MSN Messenger).*

1. Introduction

The advance of networked computers has posed a new challenge to the field of software engineering; that is, developing multi-user, computer-based applications (known as *groupware*) that support collaborative work of a group of people. Whilst single-user applications are developed to support sole users' taskwork—the work of completing a task, groupware are designed to support both taskwork and teamwork—"the work of working together" [11]. Because of these diverse goals that groupware needs to achieve, developing groupware is a challenging and complex activity as it involves multiple disciplines addressing various challenges of both technical and social requirements [10]. One challenge amongst them is to provide support for *group awareness*.

In the past two decades, supporting group awareness has been recognised by the CSCW (Computer-supported Cooperative Work) community as an important factor of successful groupware. Group awareness has been widely known as "an understanding of the activities of others, which provides a context for your own activity" [6]. Research shows that some degree of group awareness is necessary for all collaborative work [6, 11, 17]. Group awareness supports group collaboration by simplifying communication, facilitating coordination and managing coupling [10, 11, 15].

Developing mechanisms to support group awareness, referred to as *awareness mechanisms*, has been recognised as an essential requirement of groupware development [2, 4, 6, 11, 17]. Many innovative awareness mechanisms have been produced. Strict-WYSIWIS (What You See Is What I See) [20], telepointers [13], multi-user scrollbars [12], radar views [12], fisheye views [26] and auditory cues [14] are just some representative examples of commonly used awareness mechanisms.

However, current awareness mechanisms are often built specifically for the hosting groupware and by actual developers of the groupware themselves. Little research examines approaches of implementing awareness mechanisms for groupware developed by third parties. The research reported here addresses this issue by investigating flexible approaches of building awareness support for groupware.

An API (Application Programming Interface) consists of a set of commonly-used and well-defined interfaces that support an integration of different software components. An API is also known as a standard technique for developing software that hides module implementation details from consumers of those modules by separating interfaces from implementation [5]. Module interfaces allow API consumers—developers who make method calls to the API—to adapt modules for a different use without having to access the source code of the modules.

APIs have been adopted widely in building single-user applications. However, much less research focuses on studying the application of APIs in building software components for existing groupware such as modules that support group awareness for commercial-off-the-shelf groupware.

This research investigates how APIs can be used to build awareness support for groupware. We discuss a *Transparent Adaptation* (TA) approach, which utilises APIs for awareness mechanism implementation. The TA approach offers two distinctive features:

- *groupware transparency:* awareness mechanism implementation is independent from groupware development; and
- *software reuse:* software modules that are developed based on the TA approach can be reused easily in building awareness support for other groupware.

We have applied the TA approach in developing awareness support for the CoWord collaborative word processor [27] and MSN Messenger (http://messenger.msn.com/). Both CoWord and MSN Messenger provide APIs that allow access and manipulation of data objects of groupware. Our goal is to develop awareness mechanisms for CoWord and MSN Messenger without the need to access the source code of the applications.

The rest of this paper is organised as follows. The next section examines three approaches that have been used in the implementation of awareness mechanisms. Then, Section 3 describes the principles of the TA approach in terms of development goals and the software architecture. Section 4 illustrates how the TA approach has been applied in developing awareness mechanisms for CoWord and MSN Messenger. Section 5 discusses some implementation issues reflecting on our experience in applying the TA approach. Finally, we summarise the research findings and contributions as well as presenting the direction of future research.

2. Approaches to awareness mechanism development

We have recognised three major approaches to the development of awareness mechanisms for groupware: an *in-house* approach, a *toolkit-based* approach, and a *transparent adaptation* approach.

2.1. In-house approach

We use the term "*in-house*" (IH) approach to refer to the first method which has been used to develop awareness mechanisms. Mechanisms built by the IH approach are specific to particular groupware and implemented by the actual developers of the groupware. Some examples of awareness mechanisms developed in the light of the IH approach include strict-WYIWIS [20], multi-user scrollbars [12] and radar views [12].

In the IH approach, awareness mechanisms are developed mainly as research prototypes and serve as research vehicles for investigating new techniques for awareness support. Thus, the IH approach is limited in addressing the issue of extending developed awareness mechanisms to other groupware. Another limitation of the IH approach is that it requires the completion of groupware first before awareness mechanisms can be built.

2.2. Toolkit-based approach

The second popular approach to building awareness mechanisms is a *toolkit-based* (TK) approach. In the TK approach, the implementation of awareness mechanisms is assisted by well-established groupware toolkits such as GroupKit [9], COAST [18], and MAUI [14]. Groupware toolkits provide a flexible development environment that helps developers build groupware in far less time than building them from scratch. Also, the toolkits include awareness-enhanced widgets (e.g., shared user interface components) that support building awareness mechanisms. Using groupware toolkits to develop awareness mechanisms is advantageous as mechanisms can be produced in a timely manner.

However, the limitation of the TK approach is that the development of awareness support is confined within the scope of the toolkits (i.e., restricted to services and components provided by the toolkits). In addition, like the IH approach, the development of awareness mechanisms in the TK approach can only proceed after groupware has been built (by the developers using toolkits).

2.3. Transparent adaptation approach

The third approach to implementing awareness support for groupware is a *transparent adaptation* (TA) approach. In the TA approach, awareness mechanisms are developed based on APIs provided by the hosting groupware. Hence, the TA approach does not require access to the source code of groupware when implementing awareness mechanisms (so it is *transparent*).

Examples of awareness mechanisms that are developed in the light of the TA approach include Tickertape [7] and QnA [1]. Tickertape is a lightweight awareness tool that is developed on top of the Elvin notification system [19]. Tickertape receives subscription information from Elvin, and presents this information in the form of a single line scrolling message window. QnA is built as a plug-in for Trillian Pro, an Instant Messaging (IM) client. QnA facilitates

IM communication by balancing users' responsiveness. QnA is developed as a DLL (Dynamic Linked Library) and is executed from inside Trillian Pro. Both Tickertape and QnA are groupware-dependent, as Tickertape relies on services provided by Elvin, and QnA relies on adaptation of Trillian Pro functionality.

Compared to the IH and TK approaches, the main advantage of using the TA approach is a separation of building groupware and building awareness mechanisms for groupware. Thus, this approach fosters the development of awareness mechanisms for existing groupware, especially commercial-off-the-shelf groupware.

Although the TA approach has been used in developing awareness mechanisms, previous research provides few details of how to apply the TA approach. This research addresses this issue by proposing an architecture for the TA approach specifically for the purpose of developing awareness mechanisms, and presents several examples of how this approach has been applied in building awareness mechanisms. In the next section, we describe development goals, and our proposed architecture for the TA approach.

3. System architecture of TA approach

3.1. TA approach development goals

The key to the TA approach is the use of APIs to intercept users' local operations and transform them into abstract forms, which are then interpreted at remote sites via the APIs. As an interaction occurs between an awareness mechanism and its adapted groupware via APIs, there is no need to access and change groupware source code.

Two major objectives of the TA approach are to: (1) *separate* building awareness mechanisms from building groupware; and (2) maintain *QoS* (Quality of Service) of groupware by retaining existing users' interaction with groupware and high performance of the groupware (e.g., instant response for local users and fast response for remote users).

3.2. Architecture of TA approach

We present a system architecture for the TA approach that is instrumental in assisting the approach to achieve the two objectives stated above. Each awareness mechanism built according to the TA approach needs to carry out two important processes: *API Adaptation* and *Message Interception* (Figure 1).

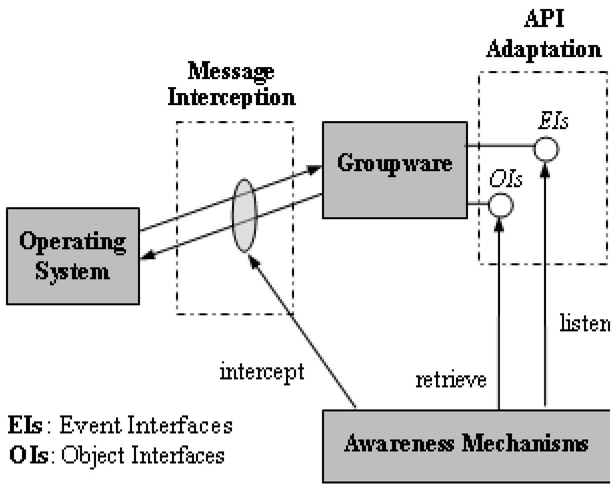


Figure 1: System architecture of the TA approach

API Adaptation: An awareness mechanism interacts with its adapted groupware via APIs. For the purpose of supporting group awareness, it is important to know evolving statuses of groupware data objects as well as users' interaction with the groupware. Thus, we categorise APIs into two groups of *Object Interfaces* and *Event Interfaces*. Object Interfaces support an awareness mechanism in accessing and manipulating *properties* of data objects in the groupware, whereas Event Interfaces allow an awareness mechanism to listen to *events* generated by users' interaction with the groupware.

Message Interception: Apart from interacting with the groupware, an awareness mechanism also intercepts messages (e.g., data message, events, etc.) sent between the Operating System (OS) and the groupware. This message interception is important as certainly not all events are supported by the groupware Event Interfaces.

3.3. Layers of an awareness mechanism

An awareness mechanism is responsible for four main tasks. First, an awareness mechanism needs to interact with its adapted groupware through APIs provided by the groupware, and intercepts event messages sent between the OS and the groupware. Second, an awareness mechanism has to process data gathered from a local site after carrying out the first task. Third, an awareness mechanism needs to communicate with another awareness mechanism at remote sites to send local data, and receive remote data of users' interactions. The final task of an awareness mechanism is to display data coming from the local

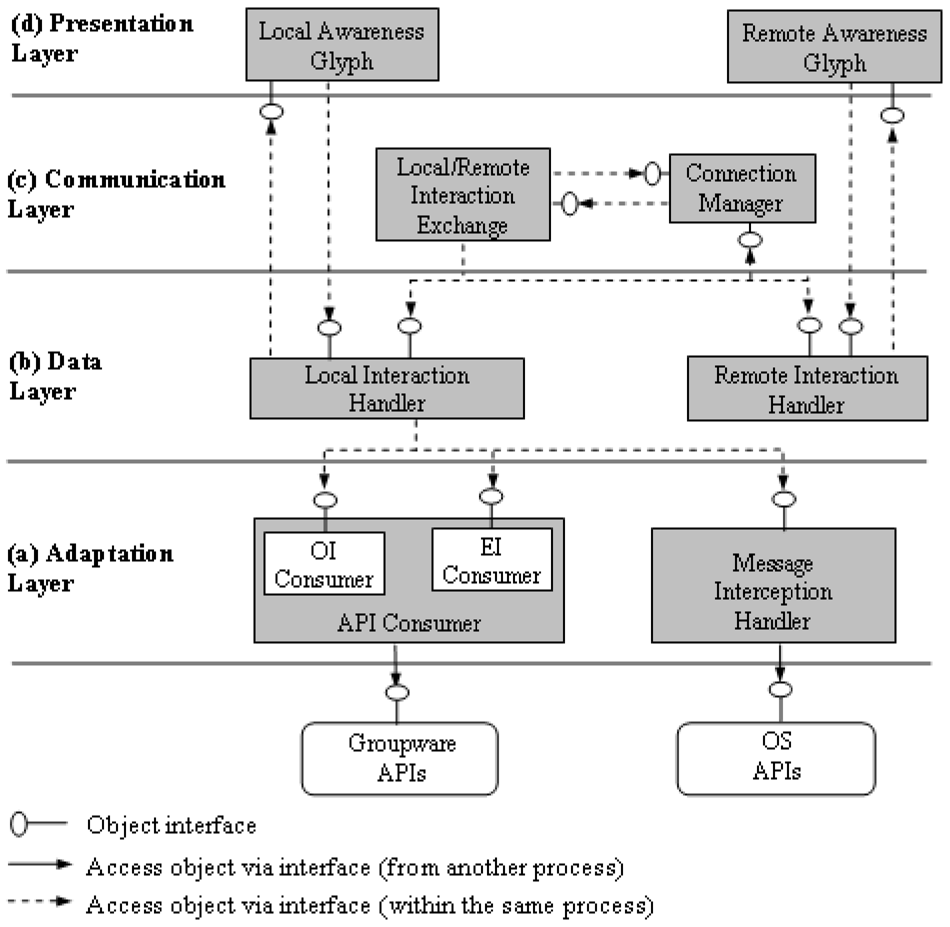

(d) Presentation Layer
Local Awareness Glyph
Remote Awareness Glyph
(c) Communication Layer
Local/Remote Interaction Exchange
Connection Manager
(b) Data Layer
Local Interaction Handler
Remote Interaction Handler
(a) Adaptation Layer
OI Consumer
EI Consumer
API Consumer
Message Interception Handler
Groupware APIs
OS APIs
Object interface
Access object via interface (from another process)
Access object via interface (within the same process)

Figure 2: Software layers of an awareness mechanism

site and remote sites in relevant forms (often as graphical representations).

To facilitate these four tasks, the component architecture of an awareness mechanism includes four layers: *Adaptation Layer*, *Data Layer*, *Communication Layer*, and *Presentation Layer*.

Adaptation Layer **(AL)**: This layer is responsible for interacting with groupware via APIs, and for intercepting messages between the groupware and the OS. AL includes two modules: *API Consumer* and *Message Interception Handler* (Figure 2a).

API Consumer is a software module that is responsible for gathering data and events from the groupware via Object Interfaces and Event Interfaces. For these purposes, API Consumer consists of two sub-modules: *Object Interface Consumer* and *Event Interface Consumer*. Message Interception Handler is responsible for intercepting event messages sent between the groupware and OS. API Consumer and Message Interception Handler are groupware-dependent, as API Consumer requires APIs to be adaptable, and Message Interception Handler intercepts only messages sent from the OS to a specific running application (i.e., the groupware).

Data Layer **(DL)**: This layer is responsible for receiving and processing awareness data of groupware objects and users' interaction. DL includes two modules: *Local Interaction Handler* and *Remote Interaction Handler* (Figure 2b). Local Interaction Handler and Remote Interaction Handler are responsible for processing data from local and remote sites (e.g., cursor movement, viewport changes, etc.), respectively.

Communication Layer **(CL)**: This layer is responsible for the communication between distributed awareness mechanisms. To support awareness in group collaboration, an awareness mechanism must provide information about both the local user's and remote users' interaction. Often, APIs do not provide interfaces for transmitting data between awareness mechanisms, as the groupware does not have knowledge of what awareness mechanisms want to exchange. Thus, CL is required. However, in the case where an awareness mechanism can utilise APIs for sending and receiving data across the sites, methods in CL can interact directly with APIs, as Object Interface Consumer does.

CL includes two modules: *Connection Manager* and *Local/Remote Interaction Exchange* (Figure 2c). Connection Manager is responsible for establishing and maintaining a connection channel between awareness mechanisms. Local/Remote Interaction Exchange is responsible for sending data of the local interaction, and receiving data of the remote interaction.

***Presentation Layer* (PL)**: This layer is responsible for presenting data gathered by DL in a selective form (often in a graphical form). PL uses different visualisation techniques (e.g., zooming) to collectively display awareness information. A separation of data (DL) and presentation (PL) is necessary to provide selective and tailorable awareness support. For example, it may occur that DLs collect the same data for two users, but PLs at the sites can show different presentations depending on different needs of users.

PL includes two modules: *Local Awareness Glyph* and *Remote Awareness Glyph* (Figure 2d). Local Awareness Glyph and Remote Awareness Glyph are responsible for displaying awareness, and information coming from local and remote users, respectively (e.g., Local Awareness Glyph shows local feedback, Remote Awareness Glyph shows remote feedthrough, etc.).

4. Applying TA approach

Using the architecture described above, we have developed awareness mechanisms for CoWord and MSN Messenger, using the C++ language. This section details how the TA approach has been applied in building awareness support. In particular, we focus on adaptation of the Word APIs and the MSN Messenger APIs for building transparent awareness mechanisms.

4.1. Adaptation of CoWord APIs

CoWord is a real-time collaborative version of the most popular single-user word processing application, Microsoft Word (referred to as *Word* for short). CoWord retains most functionality and familiar user interfaces of Word, but provides very little support for

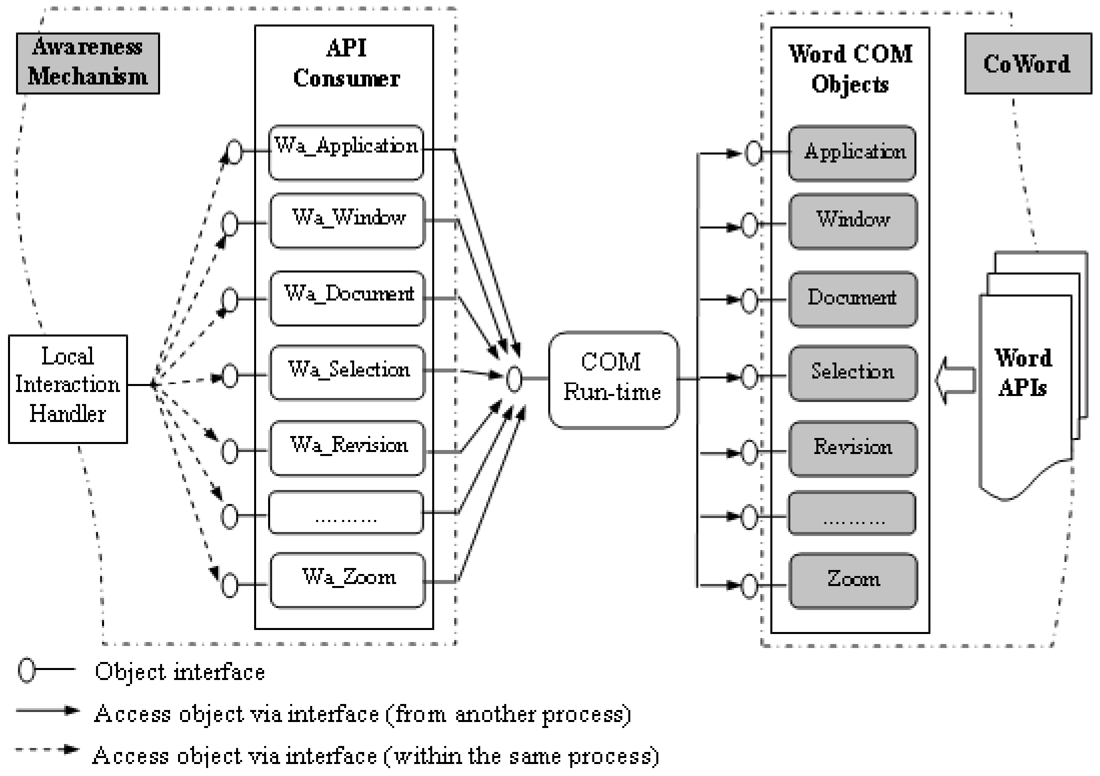


Figure 3: Adaptation of Word APIs using COM

group awareness [27]. Word provides comprehensive APIs including objects, methods, and events, which conform to Component Object Model (COM) Automation. These APIs allow developers to access and manipulate data objects such as text, tables, images, etc. in a Word document [16].

Figure 3 illustrates our solution of adapting the Word APIs for developing awareness mechanisms. The Word COM objects such as *Application*, *Document*, *Selection*, etc. expose their interfaces that allow our awareness mechanisms to access and manipulate the objects' properties. The API Consumer component of an awareness mechanism is an *ActiveX* client[1] that accesses the Word COM objects using the *IDispatch* interface[2] provided by the objects.

As shown in Figure 3, an awareness mechanism and Word are two independent applications, and communicate with one another using the COM technology. API Consumer creates a list of references to object interfaces corresponding to a list of Word COM objects. For example, a *Wa_Application* object in API Consumer maps onto an *Application* object in Word. The term "*Wa*" in front of each object's name stands for "*Word adapted*".

Within the scope of an awareness mechanism, the Local Interaction Handler component can access,

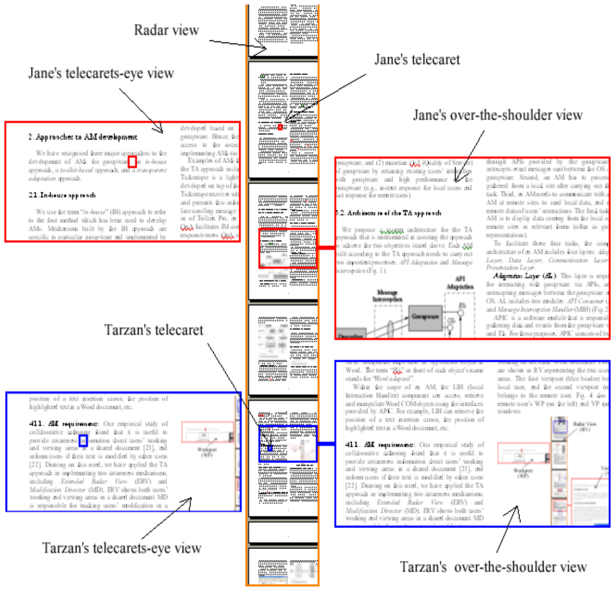

Radar view
Jane's telecarets-eye view
Jane's telecaret
Jane's over-the-shoulder view
Tarzan's telecaret
Tarzan's telecarets-eye view
Tarzan's over-the-shoulder view

Figure 4: Extended Radar View

[1] An *ActiveX* client is a software component that accesses other components' features via the other components' exposed interfaces.

[2] The *IDispatch* interface includes functions that allow API Consumer to access the methods and properties of the Word COM objects (e.g., Application, Document, etc.).

retrieve and manipulate Word COM objects using the interfaces provided by API Consumer. For example, Local Interaction Handler can retrieve the position of a text insertion cursor, the position of highlighted text in a Word document, etc.

4.1.1. Awareness mechanism requirements

Our empirical study of collaborative authoring found that it is useful to provide awareness information about users' working and viewing areas in a shared document [23], and inform users if their text is modified by other users [22]. Drawing on this need, we have applied the TA approach in implementing two awareness mechanisms, including *Extended Radar View* (ERV) and *Modification Director* (MD). ERV shows both users' working and viewing areas in a shared document. MD is responsible for tracking users' modification in a shared document.

4.1.2. Extended Radar View (ERV)

ERV [25] is an awareness tool that provides users with information about other users' viewing and working areas in a shared document. ERV includes three main windows: the *telecarets-eye view*, the *radar view*, and the *over-the-shoulder view* (Figure 4).

The telecarets-eye view shows an area in a shared document on which a user is *working*. It displays an area around a remote user's text insertion cursor (i.e., telecaret). The radar view [12] shows a miniature view of a shared Word document. The radar view also displays users' viewports, and their corresponding insertion cursors. The over-the-shoulder view displays an area in a shared document at which a user is *looking*.

These three views can be set to visible or invisible for each user. Often, the telecarets-eye view and the over-the-shoulder view of a local user are set to invisible as the local user knows where he/she is working and looking. However, the local user's viewport and the local user's text insertion cursor are still shown on the radar view. They help the local user to be aware of his/her working and viewing locations relative to those of other users.

Figure 4 illustrates the scenario when two users, Jane and Tarzan's, are working on the same Word document. Jane's telecarets-eye view and Tarzan's telecarets-eye view are shown on the left hand side; the radar view is in the middle; Jane's over-the-shoulder view and Tarzan's over-the-shoulder view are shown on the right hand side. In addition, the locations of two users' viewports are shown in the radar view.

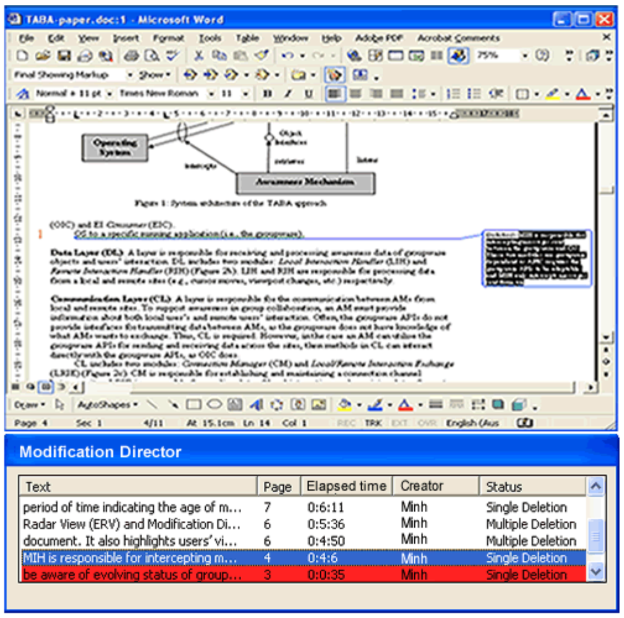
4.1.3. Modification Director (MD)

MD [25] provides group awareness by notifying users instantly when their work is modified by other users. In a Word document, modification of text includes two primitive operations: *deleting* existing text and *inserting* new text (a *replace* operation is a composite of delete and insert); and other content-*formatting* operations such as changing font size, colours, aligning text, and so on. The Word APIs provides comprehensive interfaces that support tracking of these modifications in a Word document.

The current version of MD only tracks a deletion operation—MD notifies a local user whenever his/her text is deleted by other users. And, the implementation of MD can easily be extended to track other types of modification such as insertion and text formation.

Whenever a user's text is deleted, the content of the text is shown in MD with a highlighted background colour. A colour fades after a period of time indicating the relative age of modification. In addition to displaying the content of a deletion, MD (Figure 5) also indicates the type of a deletion (e.g., single deletion, or multiple deletions), the location of a deletion in a shared Word document (e.g., page 6), and the elapsed time of a deletion (e.g., 4 minutes 50 seconds ago).

MD also allows a user to view the corresponding location of a deletion in a shared document. When a user clicks on a deletion on MD, the corresponding deletion is highlighted in a reviewing pane of the Word document.


TABA-paper.doc:1 - Microsoft Word
File Edit View Insert Format Tools Table Window Help Adobe PDF Acrobat Comments
Page 4 Sec 1 4/11 At 15.1cm Ln 14 Col 1 REC TRK EXT OVR English (Aus
Modification Director

Text	Page	Elapsed time	Creator	Status
period of time indicating the age of m…	7	0:6:11	Minh	Single Deletion
Radar View (ERV) and Modification Di…	6	0:5:36	Minh	Multiple Deletion
document. It also highlights users' vi…	6	0:4:50	Minh	Multiple Deletion
MIH is responsible for intercepting m…	4	0:4:6	Minh	Single Deletion
be aware of evolving status of group…	3	0:0:35	Minh	Single Deletion

Figure 5: Modification Director

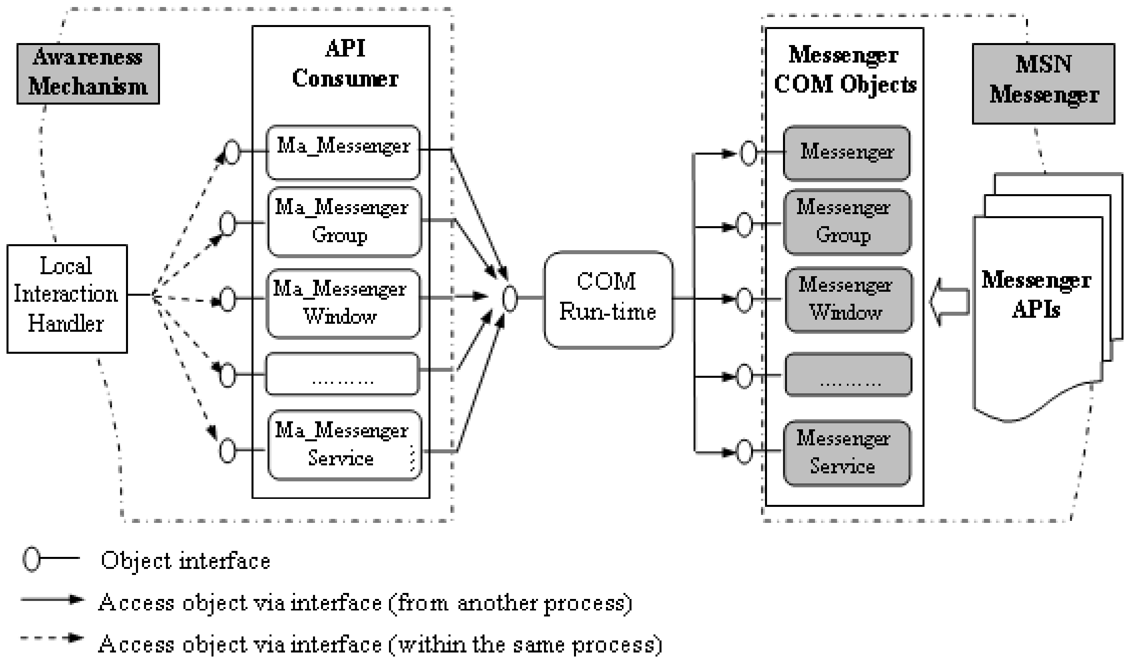


Figure 6: Adaptation of Messenger APIs using COM

4.2. Adaptation of MSN Messenger APIs

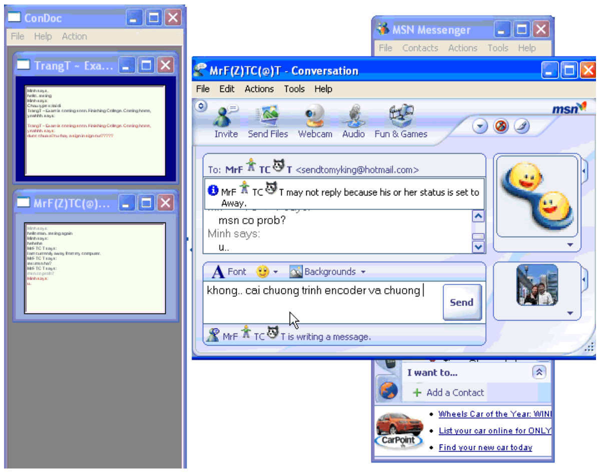
MSN Messenger is an IM client that supports semi-synchronous communication. MSN Messenger uses the Windows Messenger APIs (in this paper, we refer to these APIs as the Messenger APIs for short). The Messenger APIs provide a set of interfaces for objects and events (e.g., *Messenger*, *MessengerWindow*, *OnSignIn*, *OnSignOut*, etc.). Compared to the Word APIs, the Messenger APIs are much less complete, but still expose standard COM Automation. Figure 6 depicts our solution of adapting the Messenger APIs, which is similar to the solution used for the Word APIs. In Figure 6, the term "*Ma*" in front of an object in API Consumer refers to "*Messenger adapted*".

4.2.1. Awareness mechanism requirements

Our empirical study of IM found that there is a need for supporting awareness of multiple IM conversations [24]. To address this need, we have implemented an awareness mechanism, called Conversation Dock (ConDoc), which utilises the fisheye view [26] to support awareness of multiple IM conversations.

4.2.2. Conversation Dock (ConDoc)

ConDoc helps users manage their conversations by placing all currently active conversations in a miniature window form (Figure 7). Users can quickly *read* a conversation in ConDoc by moving a mouse over it. As a user moves the mouse over a particular conversation, the window of that conversation is enlarged while windows of other conversations in ConDoc remain unchanged. This creates a visual effect of magnification lenses described in [3, 8]. If users want to *type* further in a particular conversation, they



To: MrF TC T <sendtomyking@hotmail.com>

MrF TC T may not reply because his or her status is set to Away.

msn co prob?

Minh says:

u..

khong.. cai chuong trinh encoder va chuong

MrF TC T is writing a message.

Figure 7: Conversation Dock (ConDoc)

can drag the conversation out of ConDoc and interact with the window as normal. When users minimise a chat window, the window is put back into ConDoc instead of being placed on the task bar.

ConDoc includes visual cues to provide awareness information about the arrival of new messages. When a new message arrives at a particular conversation in ConDoc, the window containing that conversation flashes, and a new message is highlighted in another colour. The window stops flashing and the colour changes to the default colour when a user attends to the conversation by moving a mouse over the window.

5. Discussion

Here, we discuss several issues reflecting on our experience of applying the TA approach in building awareness support for CoWord and MSN Messenger.

5.1. Lack of interfaces for events

Both Word and MSN Messenger provide APIs that allow other software components to access and manipulate objects of the applications. However, these APIs are still limited in terms of providing accessible programming interfaces for events including application-driven events (e.g., window closure) and user-driven events (e.g., scrollbar movement). To address this limitation, we have to intercept messages (e.g., mouse actions, key strokes) before they reach CoWord and MSN Messenger using Win32 APIs [19].

For example, ConDoc intercepts an event of minimising a chat window (i.e. WM_MINMAXMSG). ConDoc blocks this event from reaching a MSN Messenger chat window. ConDoc intercepts and manipulates this event in such a way so that whenever a user minimises a chat window, the window is placed on ConDoc instead of the Windows Taskbar.

As another example, the lack of interfaces for events is experienced in the implementation of MD. The Word APIs provide the *Revision* object that supports tracking changes in a Word document. MD uses *Revision* to check if other users delete a local user's text. Unfortunately, the Word APIs provide no event notifying if *Revision* is modified. As a result, we have to access *Revision* periodically to check if the object has been modified.

5.2. Stability

Stability has been well-known as an important aspect of an API. A stable API should not be changed frequently. The stability of an API helps ensure correctness and independency between API consumers and the API. In the implementation of awareness mechanisms for CoWord, we found that our awareness mechanisms only work for one version of Word (i.e., the Word XP version) due to changes in APIs of different versions of Word. Our awareness mechanisms need to be modified to suit each version of Word, and it may require multiple versions of an awareness mechanism in order to be compatible with different versions of Word. Although the modification can be managed, it should be avoided in the first place by maintaining the stability of the Word APIs.

6. Conclusions and future work

This paper has discussed a *transparent adaptation* (TA) approach to the implementation of awareness mechanisms for groupware. The key to this approach is to separate the development of awareness mechanisms from the development of groupware. Awareness mechanisms are built based on the APIs (Application Programming Interfaces). We propose an architecture for the TA approach and the software design of awareness mechanisms.

To build a transparent awareness mechanism (i.e., without accessing the source code of groupware), an awareness mechanism needs to have two components: *API Adaptation* and *Message Interception*. The API Adaptation component is responsible for interaction between an awareness mechanism and its adapted groupware via the APIs. The Message Interception component is responsible for intercepting message events between the OS (Operating System) and the groupware.

We have applied the TA approach in building awareness mechanisms for CoWord—a real-time collaborative Microsoft Word—and MSN Messenger. The development of awareness mechanisms exploits the Word and Messenger APIs, hence, it does not require changes to the source code of CoWord and MSN Messenger. From our experience with the TA approach, we have found the approach advantageous in building awareness mechanisms for two reasons: (1) *groupware transparency*: awareness mechanisms are developed independent from the groupware implementation; and (2) *software reuse*: software components developed in this approach can be reused in more than one awareness mechanism.

However, because awareness mechanisms developed using the TA approach rely on APIs, we have found several development issues of using this approach. For example, the TA approach requires groupware to provide comprehensive APIs for both objects and events, and APIs need to be stable to ensure the compatibility of awareness mechanisms.

As future work, we will investigate the application of the TA approach in implementing awareness mechanisms for other groupware, and conduct usability studies to evaluate the awareness mechanisms' usefulness in supporting group collaboration.

7. Acknowledgements

This research was supported by Smart Internet Technology CRC (SITCRC), Australia. We would like to thank Nghia Xuan Nguyen and John Gheybi for their efforts in implementing the mechanisms.

8. References

[1] Avrahami, D. and Hudson, S. E., "QnA: augmenting an instant messaging client to balance user responsiveness and performance", *Proceedings of the 2004 ACM conference on Computer supported cooperative work (CSCW '04),* Chicago, Illinois, USA, pp. 515-518, 2004.

[2] Benford, S., Bowers, J., Fahlen, L. E., Greenhalgh, C., and Snowdon, D., "User Embodiment in Collaborative Virtual Environments", *Proceedings of the ACM Conference on Computer-Human Interaction CHI'95,* Denver, Colorado, USA, pp. 242-249, 1995.

[3] Bier, E. A., Stone, M. C., Pier, K., Fishkin, K., Baudel, T., Conway, M., Buxton, W., and DeRose, T., "Toolglass and Magic Lenses: the See-Through Interface", *Proceedings of the CHI '94 Conference Companion on Human Factors in Computing Systems,* Boston, Massachusetts, pp. 445-446, 1994.

[4] Carroll, J. M., Neale, D. C., Isenhour, P. L., Rosson, M. B., and McCrickard, S. D., Notification and Awareness: Synchronizing Task-oriented Collaborative Activity, *International Journal of Human-Computer Studies*, vol. 58, pp. 605-632, 2003.

[5] de Souza, C. R. B., Redmiles, D., Cheng, L.-T., Millen, D., and Patterson, J., "How a good software practice thwarts collaboration: the multiple roles of APIs in software development", *Proceedings of the 12th ACM SIGSOFT twelfth international symposium on Foundations of software engineering (SIGSOFT '04/FSE-12),* Newport Beach, CA, USA, pp. 221-230, 2004.

[6] Dourish, P. and Bellotti, V., "Awareness and Coordination in Shared Workspaces", *Proceedings of the ACM Conference on Computer Supported Cooperative Work CSCW '92,* Toronto, Canada, pp. 107-114, 1992.

[7] Fitzpatrick, G. A., Parsowith, S., Segall, B., and Kaplan, S., "Tickertape: awareness in a single line", *Proceedings of the ACM Conference on CHI 98 Summary: human factors in computing systems,* Los Angeles, California, US, pp. 281-282, 1998.

[8] Greenberg, S., Gutwin, C., and Cockburn, A., "Using Distortion-oriented Displays to Support Workspace Awareness", *People and Computer XI (Proceedings of the HCI'96),* Imperial College, London, pp. 229-314, 1996.

[9] Greenberg, S. and Roseman, M. Groupware Toolkits for Synchronous Work. In: *Computer-Supported Cooperative Work, Trends in Software Series*, ed. Beaudouin-Lafon, M. John Wiley & Sons, 1999.pp. 135-168.

[10] Grudin, J., Groupware and Social Dynamics: Eight Challenges for Developers, *Communications of the ACM*, vol. 37, pp. 92-105, 1994.

[11] Gutwin, C. and Greenberg, S., A Descriptive Framework of Workspace Awareness for Real-Time Groupware, *Computer Supported Cooperative Work, The Journal of Collaborative Computing*, vol. 11, pp. 411-446, 2002.

[12] Gutwin, C., Roseman, M., and Greenberg, S., "A Usability Study of Awareness Widgets in a Shared Workspace Groupware System", *Proceedings of the ACM Conference on Computer Supported Cooperative Work CSCW'96,* Boston, Massachusetts, pp. 258-267, 1996.

[13] Hayne, S., Pendergast, M., and Greenberg, S., Implementing Gesturing with Cursors in Group Support Systems, *Journal of Management Information Systems*, vol. 10, pp. 42-61, 1994.

[14] Hill, J. and Gutwin, C., "Awareness Support in a Groupware Widget Toolkit", *Proceedings of the 2003 ACM Conference on Group Work Group'03,* Sanibel Island, FL, pp. 258-267, 2003.

[15] Mark, G. and Prinz, W., "What happened to our Document in the Shared Workspace? The Need for Groupware Conventions", *Proceedings of the Sixth IFIP International Conference on Human-Computer Interaction INTERACT 97,* Sydney, Australia, pp. 412-420, 1997.

[16] Microsoft Corporation. http://msdn.microsoft.com/, Posted 2005, Accessed October 2005.

[17] Rodden, T., "Population the Application: A Model of Awareness for Cooperative Applications", *Proceedings of the ACM Conference on Computer Supported Cooperative Work CSCW'96,* Boston, Massachusetts, pp. 87-96, 1996.

[18] Schuckmann, C., Kirchner, L., Schummer, J., and Jorg, M. H., "Designing object-oriented synchronous groupware with COAST", *Proceedings of the 1996 ACM conference on Computer supported cooperative work CSCW '96,* Boston, Massachusetts, United States, pp. 30-38, 1996.

[19] Segall, B. and Arnold, D., "Elvin has left the building: A publish/subscribe notification service with quenching", *Proceedings AUUG'97,* Brisbane, Australia, 1997.

[20] Stefik, M., Bobrow, D. G., Foster, G., Lanning, S., and Tatar, D., WYSIWIS Revised: Early Experiences with Multiuser Interfaces, *ACM Transactions on Office Information Systems*, vol. 5, pp. 147-167, 1987.

[21] Sun, C. and Chen, D., Consistency maintenance in real-time collaborative graphics editing systems, *ACM Transactions on Computer-Human Interaction*, vol. 9, pp. 1-41, 2002.

[22] Tran, M. H., Raikundalia, G. K., and Yang, Y., "Methodologies and Mechanism Design in Group Awareness Support for Internet-based Real-time Distributed Collaboration", *Proceedings of the Fifth Asia Pacific Web Conference APWeb'03,* Xi'an, China, pp. 357-369, 2003.

[23] Tran, M. H., Raikundalia, G. K., and Yang, Y., "What are you looking at? Newest findings from an empirical study of group awareness", *Proceedings of the Sixth Asia Pacific Computer Human Interaction APCHI'04,* Rotorua, New Zealand, pp. 491-500, 2004.

[24] Tran, M. H., Yang, Y., and Raikundalia, G. K., Consumption of Multiple Concurrent Identities: the Need from the Instant Messaging Virtual Community, *The Australasian Journal of Information Systems*, vol. Special Issue, pp. 4-20, 2004.

[25] Tran, M. H., Yang, Y., and Raikundalia, G. K., "Extended Radar View and Modification Director: Awareness Mechanisms for Synchronous Collaborative Authoring", *Proceedings of the Seventh Australasian User Interface Conference AUIC'06,* Hobart, Tasmania, pp. 45-52, 2006.

[26] Weir, P. and Cockburn, A., "Distortion-oriented Workspace Awareness in DOME", *British Computer Society Conference on Human-Computer Interaction,* Sheffield Hallam University, Sheffield, pp. 239-252, 1998.

[27] Xia, S., Sun, D., Sun, C., Chen, D., and Shen, H., "Leveraging Single-user Applications for Multi-user Applications: the CoWord Approach", *Proceedings of the 2004 ACM conference on Computer supported cooperative work CSCW'04,* Chicago, Illinois, USA, pp. 162-171, 2004.