DiscoTech: A Toolkit for Handling User Level Disconnection Problems in Synchronous Groupware

Banani Roy
Queen's University
School of Computing
Kingston, ON, K7L 3N6

Nicholas Graham
Queen's University
School of Computing
Kingston, ON, K7L 3N6

Carl Gutwin
University of Saskatchewan
110 Science Place
Saskatoon, SK, S7N 5C9

ABSTRACT
During a collaboration session in synchronous groupware participants can often get disconnected, which causes various user level problems, such as interpretation difficulties, confusion and misunderstanding To date, no toolkit exists for providing programming support for developers to solve user level disconnection problems. In this paper, we present a toolkit called DiscoTech that offers a gentle learning curve programming solution to the groupware developers for handling user level disconnection problems.

Keywords
Groupware, disconnection behaviour, feedback, toolkit, plug-ins.

1. INTRODUCTION
During a collaboration session, participants get disconnected in synchronous groupware for several reasons including power failure, network outage, network latency, and explicit departure [4]. A disconnected participant loses track of the collaborative workflow causing various user-level problems, such as interpretation difficulties, confusion, and misunderstanding [4]. Effective feedback following a reconnection would help the user 'catch up' with the ongoing collaboration activities. For example, during a group chat session, suppose a participant gets disconnected for one minute due to network outage. Upon reconnection, the user sees a quick replay of the messages exchanged in her absence, allowing her to 'catch up' with the current conversation.

However, despite the promise of improved group coordination and collaboration, synchronous groupware does not yet resolve these user level disconnection problems. Addressing user level disconnection problems from the scratch adds new programming tasks to the groupware developers and increases complexity of groupware applications. It distracts the developers from constructing the application itself. In some cases, developers simply may not have considered the problem or they focused mainly on reestablishing the network connection and the consistency maintenance of the shared objects [1, 2, 3, 7]. To date, no programming toolkits exist that address disconnection considering user–level problems. The only framework that considers this issue, Disco [4], does not deeply investigate how their proposed approach would work in practice.

In order to address the user-level disconnection problems, we have built a toolkit called DiscoTech which offers a flexible, yet simple APIs that allow developers to create disconnection aware synchronous groupware applications in a simple and flexible manner.

DiscoTech is designed using the plug-in architecture proposed by Gutwin et al. [4]. These plug-ins aim to provide flexibility to the developers to adapt to various disconnection behaviours in the groupware applications. Disconnection behaviours are obtained by storing information and presenting this information (we also term it reconnection information) to the reconnected users. Compactor plug-ins reduce the size of the stored messages to optimize message delivery requirements [3]. To adapt the presentation disconnection behaviours, replayer plug-ins represent the stored messages to the reconnected users in some fashions.

DiscoTech aims to provide a gentle learning curve for the groupware developers who want to construct a groupware application with the reconnection feedback mechanisms. We term such an application as disconnection aware. For this, DiscoTech offers three levels of plug-ins solutions. With these three levels, developers can engage with the toolkit progressively to adapt disconnections behaviours with increased expressiveness.

We plan to evaluate DiscoTech in two steps. First, we will measure the expressiveness of the plug-ins using a four-dimensional design space. The design space exposes all possible strategies for presenting reconnection information for a wide range of groupware applications. We will fit the plug-ins on the design space and investigate the situation where they fail. In case of failure, our plan is to either build a new plug-in (if possible) or to mark the failure as a limitation of the toolkit. While measuring the expressiveness, we will show how the developers can engage with the toolkit progressively for adapting disconnection behaviours with increased expressiveness. Second, we will illustrate how the plug-ins work in practice by constructing various groupware applications using DiscoTech.

2. RELATED WORK
To date, Disco [4] is the only framework for handling disconnection considering user-level problems. With Disco, Gutwin et al. show that it is important to address user-level disconnection problems in synchronous groupware in order to achieve the usability of synchronous groupware. Disco handles several types of disconnections in synchronous groupware considering how disconnections are identified, what senders and receivers should do during an absence, and what should be done with accumulated data upon reconnection for three different toy applications. Disco uses the plugin architecture concept for

Conference'10, Month 1–2, 2010, City, State, Country.

supporting various disconnection behaviours. However, Disco did provide details of a practical application. It also did not address the issue of developer's progressive engagement with the framework. Using our toolkit, we attempt to address these shortcomings of Disco.

Other than Disco few tools do exist to address disconnection issues. These tools mainly reestablishing the network connection automatically and maintain the consistency of a shared object. Ensemble [1] considers developers' effort while supporting fault tolerance in groupware applications. Corona [5] and DISCIPLE [6] allow disconnected mobile clients work offline and concern about consistency maintenance of the shared object upon a reconnection. YCab [2] offers a minimal learning curve for the developers to build fault tolerant collaborative applications in the mobile environment. WebArrow [7] addresses bringing a disconnected client into an operational state using a complex distributed algorithm. None of these tools, however, are intended to provide feedback to the reconnected users for solving the user-level disconnection problems, which is the main goal of our DiscoTech toolkit.

3. DESIGN PRINCIPLES OF DiscoTech

Due to disconnection in synchronous groupware several user level problems occur such as interpretation difficulties, confusion and misunderstanding. To overcome the user level problems, it is important to provide effective feedback to a reconnected client to catch up with the current collaboration activities. For example consider the following two scenarios where feedback helps:

Scenario 1: *Bonny, a PhD student, is in a chat session with her two supervisors, Clive and Nett using a chat application. When someone types, the client applications display the 'keypress' event instantly in the chat-viewing window to provide typing awareness. When someone presses 'ENTER', the applications display the chat messages while removing the 'keypress' awareness messages.*

However, when Bonny types her study update of the last week, Clive gets disconnected due to network outage. Despite Clive's disconnection, Bonny and Nett continue chatting. The disconnection tools integrated with the chat application stores all the chat messages sent from Bonny and Nett while discarding the 'keypress' awareness messages because upon a reconnection these messages are not important for Clive.

When Clive rejoins to the chat session, the disconnection handling tools display the stored chat messages in his chat-viewing window one after another with a short delay between them. After reading the messages, Clive gets an idea about his student's progress and starts typing his next message without having any confusion that could arise if he would not see the missed messages.

Scenario 2: *A group of people use a shared drawing editor to collaboratively draw a UML diagram. However, during the ongoing collaborative drawing session, Sally, a group member, gets disconnected for two minutes. The disconnection toolkit adapted with the groupware application stores all the drawing events of the last 50 seconds while discarding the rest due to the limited bandwidth.*

After rejoining to the drawing session, she sees the replay of the drawing activities of the last 50 seconds at a double speed of the original on her drawing window, which helps her understand how changes occurred in the UML diagram. If Sally could not able to see the replay, she would have been confused about the changes.

In our toolkit, we have used the idea of providing feedback to the reconnected users for solving the user-level disconnection problems. For example, in scenario 1, Clive gets feedback by seeing the missed messages and overcomes his confusion about his student's progress, and in scenario 2, Sally gets feedback by seeing the replay of the drawing activities and overcomes her confusion about the changes in the UML diagram. In this paper, we call a groupware application disconnection aware when it provides such feedback to the reconnected users.

4. ARCHITECTURE FOR FEEDBACK

The main requirement for our toolkit is to provide feedback to the reconnected users. To satisfy the requirement, the toolkit needs to adapt various disconnection behaviours into the groupware applications. For example, in scenario 1, the feedback of the chat application characterizes two disconnection behaviours: (1) discarding the awareness messages, and (2) flashing the chat messages with a short delay between them. As such, there are a wide range of disconnection behaviours for various groupware applications. In order to adapt the various disconnection behaviours, we have exploited the plug-in architecture (proposed by Gutwin et al.) as the underlying architecture of DiscoTech. These plug-ins are intended to adapt a wide variety of the disconnection behaviours.

In Figure 1, we show the architecture of a client/server based disconnection aware groupware application with DiscoTech. In normal operation, the client application generates a sequence of *events* which are processed locally and sent to the server that multicasts the *event*s to the other clients. There is a clean separation between the application and DiscoTech's components. The application only interacts with the *DiscoWrapper* component (that works as a mediator between the application and DiscoTech's components). For this feature, DiscoTech's core components are reusable across various groupware applications.

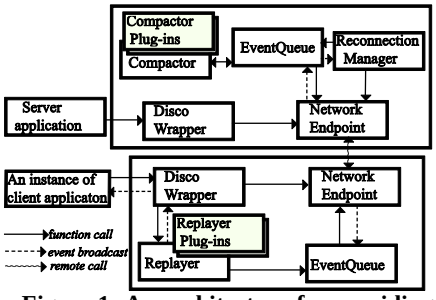

Figure 1: An architecture for providing feedback

All the complexities needed to handle disconnection and passing messages over the network are embedded in DiscoTech's components, and developers do not need to expose them. Therefore, at the simplest level, the developers need to learn simple APIs to pass events through *DiscoWrapper* while the rest is taken care of by DiscoTech. In the following, we explain how DiscoTech takes care of a disconnection situation and provides feedback to the reconnected user by using the compactor and replayer plug-ins.

4.1 Compactor Plug-ins

The server side *Event Queue* (Figure 1) component stores messages in the event queue in preparation for the eventuality of disconnection. Different disconnection behaviours can be obtained based on what portion of messages will remain in the storage. For example, in scenario 1, the chat application stores the chat messages while discarding the unimportant awareness messages and in scenario 2, the drawing editor stores the messages of the last 50 seconds discarding drawing events older than that due to limited bandwidth. In these scenarios, *Event Queue* behaviours are adapted based on the applications' event delivery/ QoS requirements [3] and its *event queue* capacity. In order to address different disconnection *Event Queue* behaviours, we have proposed compactor plug-ins intended to reduce the size of the *event queue* for satisfying the event delivery requirements.

The *compactor* component (Figure 1) can easily incorporate a compactor plug-in using the inheritance feature of object oriented programming language, where the compactor component offers a base class and the plug-ins are the sub classes overriding its *compact* method. A compactor plug-in can be built using a simple pattern where it will access the event queue periodically and will reduce its size either by eliminating some events or transforming them into another representation

4.2 Replayer Plug-ins

When a client is reconnected, the *ReconnectionManager* (Figure 1) sends the processed stored events via the *NetworkEndPoint*. The client side *NetworkEndPoint* receives the events and stores them in the event queue. *Replayer* retrieves them from the *event queue* and compares the timestamp of the events with current time; if the time interval is higher than a given *threshold* value, the *Replayer* identifies that these events are for providing feedback to the reconnected users.

Different disconnection behaviours can be obtained based on the fashions the reconnection messages will be presented to the reconnected users. For example, in scenario 1, the chat application presents the chat messages by flashing them one after another with a quick delay between them, and in scenario 2, the shared drawing editor replays the messages at a double speed. To address the various presentation behaviours, we have proposed replayer plug-ins. *Replayer* calls a replayer plug-in to adapt the presentation behaviour. The replayer plug-in processes the events (e.g. the double speed replayer plug-in compresses the time interval of the events for replaying them at a double speed) and sends them to *DiscoWrapper*, which passes them to the client application that displays the events to the reconnected user as feedback.

5. THREE LEVELS OF SOLUTIONS

With this toolkit, we offer three levels of plug-ins solutions for adapting a wide range of disconnection behaviours in the groupware applications. By going through the three levels developers can engage with the toolkit step by step and can adapt disconnection behaviours with increased expressiveness. Thus, DiscoTech offers a gentle learning for the developers who want to construct the disconnection aware groupware application.

5.1 Generic Plug-ins

These plug-ins can be used without any customization. For example, the compactor plug-in that discards events older than 30 seconds is generic because it can be applied to various groupware applications without knowing the internal event structure. DiscoTech provides the generic plug-ins for supporting simple disconnection behaviours, e.g. replaying the last 10 events or replaying the last 50 seconds of events at a double speed.

Developers can use start by using generic plug-ins without the need to learn anything difficult. They simply need to learn how to interact with the *DiscoWrapper* component (Figure 1) to send and receive events over the network and this can be done with simple APIs. The benefit of this approach is that with a very little effort they will get some disconnection behaviours built into their application. Generic plug-ins may be sufficient for many groupware applications.

5.2 Partial Generic Plug-ins

If developers want to add complexity, they can use the partially generic plug-ins. These compactors require some customization. DiscoTech offers the basic template for writing a plug-in, but the developers need to customize a specific method for adapting the plug-in in their applications.

For example, in an *Aggregating* compactor the *event queue* is divided into two-second intervals, and all events within that interval are aggregated into a single event. The basic compaction algorithm is generic, but the specifics of how a set of events is aggregated into a single event are application-specific. For example, a telepointer application uses *averaging* (a group of events is combined to a single event whose position is the average of the group**),** whereas a chat application uses *chunking* (a sequence of single-character events is combined to a single event with a sequence of characters). Here the telepointer and the chat applications use different methods for combining the events. To address such disconnection behaviours, DiscoTech offers the following API for the *Aggregating* compactor:

```
abstract class AggregatingCompactor : Compactor
    {
      void Compact();
      virtual Event Aggregate(List<Event>);
    }
```

The developers have to fill the virtual *Aggregate* method that requires them to learn the event structure of DiscoTech. Therefore, by using the partial-generic plug-ins developers can expose a wide range of disconnection behaviors while engaging with the toolkit one step further than the generic ones.

5.3 Application-specific Plug-ins

For some cases groupware applications might need to have application-specific disconnection behaviours. For example, a telepointer application can use a *TelepointerTraces* plug-in to visually represent the information about the recent telepointer positions on the current frame. Although in such cases the constructions of plug-ins require more engagement with the toolkit than the other two plug-ins levels, the underlying plug-in architecture of DiscoTech offers flexibility to do so. Developers only have to concentrate on their plug-ins algorithms and not on their adaptation technique with the toolkit. As a result, they can expose a wider range of disconnection behaviours.

6. DESIGN SPACE FOR PLUG-INS

The goal of DiscoTech is to ensure the feedback for the reconnected users for a wide range of groupware applications. For this, DiscoTech should be able to adapt required disconnection behaviours into the groupware applications. In so doing, a library of plug-ins is required. We need to analyze how

the plug-ins fit for different groupware disconnections behaviours. For the analysis, we have defined a four-axis design space (Figure 2). The goal of design space is to capture all possible strategies for presenting reconnection information, which would allow us to identify possible disconnection behaviours of various groupware applications. Each axis in the design space ranges over three values and each point in the design space is a strategy. If our plug-ins library fails to cover any strategy, we will either build a new one (if possible) to cover it or we will mark the failure as a limitation of our toolkit. In the following, we discuss how some strategies fit into each axis of design space.

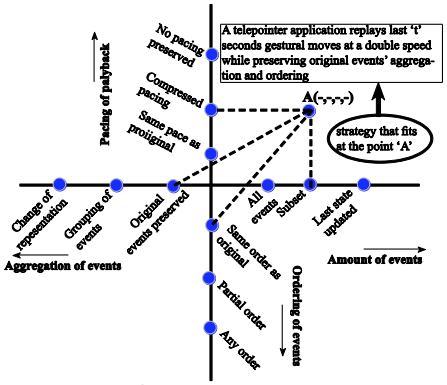

Figure 2: A four dimensional/axes design space

Amount of events: specifies what portion of backlogged events needs to be delivered. The first value of this axis is *all events replayed* where an application would replay all the accumulated information, e.g. a chat application might replay all the messages when a user is reconnected after a few seconds. The second value is the *subset of events replayed* where an application replays events after eliminating some events based on the factors, such as the age of the events, number of events and groupware message types (such as awareness and transactions [3]). For example, in scenario 2, the strategy was to replay the last 50 seconds of drawing events. The third value is the *last state updated* where only the latest event will be replayed, e.g. a power point presentation application presents the latest slide.

Pacing of playback: refers to how closely the pace of playback needs to be matched with the original. The first value of this axis is the *same pacing as original*, e.g. a telepointer application replays the line drawing events at the same pace as the original while discarding the telepointer awareness messages. The second value is the *same proportional pacing with compressed timeline* where events are replayed after compressing their time intervals, e.g. a video application replays a video stream at a double speed. And the third value is *pacing not preserved* where events are replayed without having any pace, e.g. an audio conferencing application presents the audio transcription in a single frame.

Aggregation of events: specifies grouping of events. The first value of this axis is the *original events preserved*, e.g. a shared editor replays drawing operations without grouping them together. The second value is the *grouping of events* where a group of events is combined into a single event, e.g. a telepointer application combines a group of telepointer positions into a single position using averaging. The third value is the *change of representation***,** where events are transformed into different representations, e.g. a telepointer application uses change of representation to heat map

Ordering of events: specifies at what order the events need to be processed. The first value is the *same order as original*, e.g. a chat application replays messages using the original order. The second value is the *partial order* where ordering of each message type would be maintained but not the overall ordering of all message types, e.g., a shared white board application replays different reliable messages [3] such as chat, annotations and session management in a partial order. The third value is simply *any order*.

7. EVALUATION

We will first illustrate how expressive the three levels of plug-ins solutions (mentioned earlier) are for supporting a wide range of disconnection behaviours. For measuring the expressiveness we will go through the design space and will analyze to what extent the three levels of plug-ins cover the design space. We will also analyze how these three levels engage developers progressively in order to adapt the disconnection behaviours with increased expressiveness.

Then**,** we will show how various compactor and replayer plug-ins work in practice for covering the design space. For this we will construct a wide variety of disconnection aware groupware applications.

8. CONCLUSION AND FUTURE PLAN

To the best of our knowledge, DiscoTech is the only toolkit for providing programming support to the developers for solving user level disconnection problems. It intends to engage a developer progressively to address disconnection behaviors with increased expressiveness.

Our future plan is to construct plug-ins to cover the design space and then investigate them extensively by constructing various disconnection groupware applications with DiscoTech. We also plan to build some plug-ins that provide feedback to the disconnected users.

9. REFERENCES

[1] Ding, Vogel, W. Object Oriented GroupWare using the Ensemble System, *Proc. OOGP*'97, pp .

[2] Buszko, D., Lee, W., and Helal, A. Decentralized adhoc groupware API and framework for mobile collaboration. *Proc. Group'01*, 5-14.

[3] Gutwin, C., Fedak, C., Watson, M., Dyck, J. and Bell, T. Improving Network Efficiency in Real-Time Groupware with General Message Compression, *Proc ACM CSCW'06*, 119-128.

[4] Gutwin, C., Graham, T.C.N., Wolfe, C., Wong, N. Alwis, D. Gone but not forgotten, *Proc ACM CSCW'10*, 179-188.

[5] Hall, R., Mathur, A., Jahanian, F., Prakash, A., and Rassmussen, C. Corona: a communication service for scalable, reliable group collaboration systems, *Proc. CSCW'96*, 140-149.

[6] Ionescu, M., Meng A.K. and Marsic, I. Dynamic Content and Offline Collaboration in Synchronous Groupware. *Proc CTS'*02.

[7] T. Graham, R. Kazman, and C. Walmsley. Agility experimentation: Practical techniques for resolving architectural tradeoffs, *Proc.* ICSE '07, 519-528.