Using Behaviour Characteristics to Improve Groupware Performance

Jeff Dyck, Carl Gutwin, and Dwight Makaroff
University of Saskatchewan
57 Campus Drive
Saskatoon, Saskatchewan, S7N 5A9
(306) 966-6593
jeff.dyck@usask.ca

ABSTRACT
Real-time distributed groupware often performs poorly when network conditions are not optimal. One approach to improving performance is to first look at the characteristics of the information being exchanged by the system, such as payload size and frequency. However, the diversity of tasks, implementations, and groups makes it is difficult to generalize about groupware characteristics. This paper suggests that constrained groupware problems can be characterized efficiently using a low cost approach to identify performance problems and suggest ways that performance can be improved. A case study to characterize GroupDraw. a shared whiteboard application, demonstrates how a constrained groupware problem can be characterized. The analysis of the extracted characteristics identifies a wide range of performance problems and suggests many possible opportunities to improve performance. The success of the case study not only validates this approach, but also provides knowledge about the characteristics of one constrained groupware situation.

General Terms
Performance, Design

Keywords
Groupware, CSCW, performance, adaptive, QoS, group events, characteristics, characterizing.

1. INTRODUCTION
Real-time distributed groupware allows remotely located people to collaborate using computers in real time. Some common examples include chat, online games, and shared whiteboards. This type of groupware is commonly cited for its poor performance (e.g. [3, 17, 19]). Frequently noted performance problems include delays, poor reliability, concurrency problems, and poor synchronization. These problems make groupware frustrating to use and can cause collaboration to break down [7].

One approach to improving performance is to first study the behaviour characteristics of a system, which are properties that may affect performance such as network traffic patterns or distributions of events generated by the system. Understanding these characteristics can isolate trouble areas and suggest solutions. Based on the success of this approach in related areas, such as real-time streaming video and real-time distributed systems (e.g. [1]), it seems worthwhile to attempt to use behaviour characteristics to help solve performance problems in groupware.

However, the behaviour characteristics of groupware are poorly understood, which makes it difficult to improve groupware performance. For example, we don't know how large we should expect groupware messages to be or how frequently messages will need to be sent. As a result, it is difficult to determine how to improve performance since we don't know what is causing problems or what we can take advantage of when developing new performance enhancing techniques. Knowledge of the behaviour characteristics of groupware is long overdue.

This paper examines what is required to extract behaviour characteristics from groupware and offers a low cost approach for characterizing a constrained groupware situation. Our theory of extracting behaviour characteristics involves three parts: first, a determination of the level of analysis to use for the extraction; second, an understanding of how quality of service requirements relate to characteristics; and third, an examination of the diversity of groupware to suggest what is required for behaviour characteristics to be meaningful. We also discuss how knowing groupware characteristics and quality of service requirements leads to performance enhancements.

A cost-effective approach for extracting groupware characteristics is then demonstrated through a case study. The case study focuses on extracting behaviour information from GroupDraw, an academic shared whiteboard application based on GroupKit [15]. The study was constrained to three different user groups, two different group tasks, and a single implementation. The method used for extracting and analyzing the characteristics of GroupDraw is described and provides an approach that could be used for other groupware situations as well. A performance analysis follows that uses both the characteristics that were extracted and the QoS requirements to identify where performance problems occur as well as how they could be resolved. We found the approach to be both efficient and thorough for identifying performance problems as well as for discovering possible ways of solving the problems.

2. RELATED WORK

A limited amount of work has been done to learn about the characteristics of events generated by groupware systems. Currently, no general information about the performance characteristics of groupware applications is available. For example, we do not know what types of statistical distributions describe groupware traffic in general or what sizes groupware message payloads are. However, some related work that will help to understand these characteristics has come out of the groupware community. There has also been a large amount of work on extracting performance characteristics in related areas such as multimedia and distributed systems.

Work on improving the performance of groupware has typically used an implementation-based approach where systems are built, user trials are run using real networks, and the performance problems are extracted. This is a very cumbersome process, and some performance studies have reduced the overhead by simulating part of the environment using network simulators and workload generators (e.g.: [2]). Zereal, a tool designed specifically for simulating Massively Multiplayer Online Games (MMOG) for performance research, has attempted to take this one step further by providing a flexible simulation system that is designed to work for a wide range of MMOG situations [5]. Bargh and ter Hofte [2] present a model for evaluating performance that defines performance qualities and metrics, which can be applied to the evaluation of performance using this type of approach. Though time consuming, implementation and evaluation has been a successful approach for measuring performance problems and isolating problem areas, but has provided fewer clues about how to fix the performance problems. In all of these approaches, implementations are built, tested, and evaluated, but the events generated by the groupware system are not recorded and analyzed directly.

There is little existing work that has focused on extracting statistical behaviour characteristics from group events or has made generalizations about how groupware applications behave. One example of this type of work was performed by Isaacs et al [9], where Instant Messaging (IM) patterns of users were gathered and analyzed to see how IM was being used in the workplace. This data was not collected with the purpose of improving the performance of IM systems, but rather to learn about how different types of workers were using IM. However, it seems feasible to be able to use this statistical work based on group event logs to look for ways to improve application performance.

Related areas such as telecommunication, multimedia, and distributed systems have produced a large amount of traffic characterization work (e.g.: [4]) with the purpose of obtaining information that can lead to performance improvements. Many approaches to extracting characteristics of networks have been developed and applied to specific domain areas. For example, Rueda and Kinsner [16] surveyed techniques available for characterizing telecommunication traffic. The knowledge gained from these characterization studies have led to both performance improvements in applications and development of new performance enhancing techniques (e.g. [1]). Groupware characterization studies could benefit from the wisdom gained in these related areas.

3. GROUP EVENTS

Characterizing groupware requires that we must first consider what to characterize, and we suggest that the characteristics of group events are more significant than those of groupware network traffic. Most characterization efforts designed to improve performance have looked at network traffic directly. This low level view is not the most suitable way to examine groupware characteristics since the information that travels over the network can be affected heavily by the way the groupware application is implemented to send the information. A more direct approach is to look at what information must be sent by the system independently of how it is being sent, as this eliminates the influence of the groupware system's network implementation on the characteristics of the information generated by the system. It is likely that network traffic characterization will become useful for fine-tuning groupware network implementations, but to get that point, we must first solve the more large scale performance problems which are best seen independently of current inefficient groupware network implementations.

Group events are produced by groupware applications whenever something needs to be sent to other users. These events typically result from a user action and are used to convey changes to group data, control information, and awareness information. For example, in an application that uses a telepointer widget, when a user moves their mouse, a group event is triggered to notify other users of the new pointer position to help them to remain aware of what the user is doing. These events result in groupware messages that need to be sent across the network to other group members.

Group events have characteristics that directly impact the network performance of a groupware system, such as the number of bytes in the message the event produces, the maximum frequency with which the event is likely to occur, and the probability of a message to be triggered given other known information. These characteristics are not currently known for groupware applications.

Group events also have characteristics that will be able to be used opportunistically when developing network performance enhancing techniques for groupware. For example, knowing what size messages are and when breaks in the flow of information are likely to occur could suggest performance enhancing techniques that specifically take advantage of these characteristics. We currently do not know what these opportunities are for groupware.

4. QUALITY OF SERVICE

Group events have quality of service (QoS) requirements that vary for each event. For instance, a telepointer event must have low delay for the information to be useful to other users, but some loss can be tolerated without any serious impact. Comparatively, a chat message can tolerate some delay but cannot tolerate loss, as the meaning of a chat conversation can be changed if messages are lost. The performance of groupware systems can be seen as the system's ability to meet the quality of service requirements of the group events. These QoS requirements are important to consider when testing the performance of groupware systems since an improvement in performance for one event may lead to a decrease in performance for another. No formal QoS model currently exists for groupware, but there are general QoS models that can be applied to groupware, such as ISO/IEC 13236 [10].

Developing performance enhancing techniques requires knowledge about the performance goals for the information being sent. These performance goals are specified in terms of quality of service (QoS) requirements. QoS and characteristics are tightly related to performance: the event characteristics define how much is sent and how often, while the QoS requirements define the performance requirements for anything that is sent. Together, these suggest ways that groupware systems can be optimized using techniques that meet QoS requirements and consider the characteristics of events. Although this paper is not about QoS, it must be considered in terms of how it affects the transition from characteristics to knowledge of performance.

Though QoS requirements are not formally defined for groupware, they can be applied reasonably by someone with expertise in groupware using good judgment. However, event characteristics must be discovered in order to take advantage of them.

5. CHARACTERIZATION CONSTRAINTS

Group event characteristics are poorly understood in general due to the diversity of groupware and because there is no low-cost method that can assist with extracting groupware characteristics. This section discusses how the diversity of groupware limits what can be characterized in general.

Real-time distributed groupware is very diverse. A wide range of applications fit into this category including shared whiteboards, network games, conferencing systems, authoring tools, and many more. Each groupware application typically supports many group tasks, and is used by a wide range of groups with varying numbers of users from different backgrounds and with different purposes. Additionally, there are many different ways of building groupware, which implies that applications that serve the same purpose may still behave in very different ways.

As a result of this diversity, it is very difficult to make generalizations about the characteristics of group events. There are three main features of groupware that make it difficult to generalize:

- Groupware supports a multitude of tasks that behave differently.
- There are few groupware implementation standards.
- Users and groups behave differently.

5.1.1 Diverse Tasks

Group tasks supported by groupware applications produce very different event characteristics. For example, a fast-paced real-time strategy game will behave very differently from a chess game or a shared whiteboard. Even a single groupware application may support many different user tasks. For example, users can flow chart a project plan, draw a picture, or draw a layout design for a room using the same shared whiteboard. Since group events result primarily from user actions, the group events will differ greatly for different tasks since user activities can vary greatly for different tasks. The wide variance in group tasks makes it very difficult to make general observations about the characteristics of group events.

Group event characteristics can however be extracted for common tasks. Specific tasks, such as having a discussion via Instant Messaging or flow charting the process for changing a tire, can be modeled and group event characteristics can be extracted for those individual tasks (e.g.: [9]). An approach to modeling tasks is offered in the case study below.

5.1.2 Few implementation standards

There are many ways to build a groupware system. As a result, the group events generated by groupware applications depend greatly on how the system generates these user events. For example, a telepointer in one groupware system may simply generate a group event at each mouse interrupt, while another groupware system will poll for mouse positions 15 times per second and generate a group event if the mouse location has changed. The differences in implementations prevent generalizations from being made about how groupware behaves since the implementation determines how group events are generated.

Characterizing groupware is therefore better done for a single application at a time. The group event characteristics for one application will not likely be the same, even for the same task performed by the same group of people. These application-specific characterizations may be able to be used by other applications that use similar implementation styles; however, the only way to guarantee accurate characterizations is to perform analyses of individual applications.

5.1.3 People are different

Groupware is used by many different people with different backgrounds and who use groupware systems differently. One example of these differences in behaviour of groupware users is described in Isaacs et al [9], where advanced Instant Messaging system users were found to have much different behaviour patterns than beginner users. Also, different groups work together in different ways: some groups may have only one or two active members at a time and others play more of an observer role (e.g.: as with teleconferencing applications), while other groups may consist of all highly active users (e.g. as seen in many network games). In some groups, members work very closely together, with rapid turn taking and sophisticated coordination mechanisms, while other groups work more autonomously and only consult back with other group members occasionally. Since group events are generated primarily from user actions, these differences in the ways that people use groupware make it difficult to generalize about the characteristics of group events.

Characterizing groupware can be done by targeting specific types of users and groups. People can be carefully modeled to represent typical types of common users and groups for a particular task or implementation.

5.1.4 An efficient approach is required

The constraints on group event characterization suggest that an efficient, low-cost approach is required. Characterization studies in other related areas have typically required a large amount of effort and take place over a long period of time. The costs associated with this type of an approach would be difficult to justify in most groupware cases considering that group event characterization can only look at a single, well-defined problem within a small scope. Therefore, a fast, inexpensive methodology for characterizing group events must be developed in order to make characterization feasible for groupware. The case study in this paper demonstrates such an approach.

6. USING EVENT CHARACTERISTICS TO IMPROVE PERFORMANCE

Event characteristics can suggest many possible ways to improve performance. There are many approaches to improving the performance of groupware that are common techniques, but are not commonly exploited in groupware. Analyzing group event characteristics can help to identify problems and opportunities that can lead to their solutions. For a simple example, applications that generate telepointer events on every mouse interrupt could benefit from combining several telepointer events into single groupware messages before sending rather than sending a message each time a mouse interrupt takes place. The messages could then be played out at the receiver using a buffering technique or using traces [8]. There are many existing techniques that can be used to improve groupware performance, and the characteristics of the group events help to guide which ones may be appropriate. Some examples of existing techniques that could be applied include: merging events, buffering, error correction, rate control, consistency management optimizations, application layer routing techniques (e.g. application layer multicast), compression, message dropping, and priority scheduling.

New techniques and customized techniques can also be developed as a result of knowing the characteristics of group events. This behaviour information can identify opportunities for optimizations, as has been demonstrated in work from related areas. As a simple example, consider two event types that usually follow each other very rapidly; combining them into one message if the interval is within a prescribed time limit would reduce the network traffic required to send those events. This is a customization of a merging technique that merges two event types using knowledge of how the events typically relate.

The QoS requirements defined for each of the event types serve as contracts for what qualities must be maintained. Any optimization selected must be verified to ensure that the technique is also suitable for meeting the QoS requirements. Additionally, QoS requirements can be exploited as a result of characteristics that become known. For example, if an event repeats very rapidly, it is possible to improve performance by dropping events if the reliability requirements and/or the refresh rate requirements allow it. Using the QoS requirements to guide finding appropriate techniques helps to ensure that QoS is maintained while improving performance.

7. CASE STUDY: GROUPDRAW

To gain experience with characterizing groupware, a case study was run using GroupDraw, a shared whiteboard application based on GroupKit [15]. The purpose of the case study was to extract group event characteristics and to use these characteristics to suggest possible ways to improve the performance of GroupDraw for certain tasks, groups, and in general if possible. This section describes the case study and presents results. The next section discusses these results and comments on what was learned about the effectiveness of the approach.

7.1 Methodology

The approach used to extract characteristics is broken into four main activities: defining the event types that the system produces; assigning QoS requirements to each event type; gathering data; and analyzing the data to extract group event characteristics. The characteristics are then evaluated to identify performance problems. Solutions to the problems consider the nature of the problems, the characteristics that can be used advantageously to solve them, while the QoS requirements set the bounds for what must be achieved.

7.1.1 Defining Event Types

The first step in characterizing group events was to extract the individual types of group events from the application. Groupware applications typically have many types of group events. For example, a shared whiteboard may define group events for mouse movements, selecting items, drawing, dragging, deleting, and changing tools. It was important to separate each of these types of events so that they could be characterized both individually and in combinations since optimizations can be made to improve the performance of both individual and combined event types.

Extracting events from GroupDraw was done using a runtime inspection, followed by a source code inspection. The runtime inspection served to identify all of the group events at a high level and was done simply by using the system and making note of each type of group event that appeared to be present. A source code analysis was then performed to confirm that the events observed at runtime were complete and identified correctly, to see how they were triggered, and to identify the locations in the source code where they were triggered. Some of the events were generated by the application and some were generated by the GroupKit toolkit. This added some complexity in identifying a complete set of locations in the code where group events were triggered.

Once all of the events had been identified, the source code was modified to enable event logging. A logging function was added to the code library. The function was optimized to keep the file open for writing until the application was terminated, which improved performance so that the logging made no noticeable difference on the application behaviour. The function took two parameters, event name and event payload, which it added to each log entry. It also added the time of the event to each log entry. An example of the log format is shown in table 3.

A call to the logging function was added in each location where a group event was generated. GroupDraw generates text formatted messages that are sent to the other users, as does the GroupKit toolkit. These text formatted messages were considered to be the payload and the text strings were passed to the logging method as the event payload. Each of the events were named using a common plain text name so that the logs would be readable, and these names were also passed in as a parameter to the logging method.

The GroupDraw event types are as follows:

- telepointer: indicates a new location for a telepointer
- addLine: adds a line to the whiteboard
- addText: adds a field that can contain a string of text to the whiteboard
- addBox: adds a box shape to the whiteboard
- addCircle: adds a circle shape to the whiteboard
- moveOrResizeObject: moves or resizes an object on the whiteboard

- modifyText: modifies the caption of a text entry on the whiteboard; note that all text fields are added as blank and then modified every time a character is added to the field

7.1.2 Assigning Quality of Service

QoS requirements were then applied to each of the event types. There is not currently a QoS model specifically for groupware, so the general QoS model defined in ISO/IEC 13236 [10] was used to guide the process. Table 1 summarizes the high level characteristics presented in ISO/IEC 13236 that were considered when assigning QoS requirements.

The QoS requirements were defined by considering how each of the event types would be realized in the application and the effects of each of the characteristics on the user and on the application. This work was primarily guided using intuition and experience since there is no formal approach to assigning QoS requirements for groupware and no guidelines that are specific to individual event types. Examples of some of the QoS requirements assigned to the event types are shown in table 2.

Table 1: A summary of ISO/IEC 13236 QoS characteristics

Time-related characteristics	Date/time
	Time delay
	Lifetime
	Freshness
Coherence characteristics	Temporal coherence
	Spatial consistency
Capacity-related characteristics	Capacity
	Throughput
	Processing capacity
	Operation loading
Integrity-related characteristics	Accuracy
Safety-related characteristics	Safety
Security-related characteristics	Protection
	Access control
	Data protection
	Confidentiality
	Authenticity
Reliability-related characteristics	Availability
	Reliability
	Fault containment
	Fault tolerance
	Maintainability
Other characteristics	Precedence

Table 2: QoS requirements for GroupDraw event types

telepointer:
freshness: 5 updates/sec minimum; 15 updates/sec maximum
time delay: max delay 100ms
reliability: max loss rate: 10%; max burst loss: 5
precedence: <1% out of order
addLine:
time delay: max delay 300ms
reliability: 100% reliable
addText:
time delay: max delay 300ms
reliability: 100% reliable
addBox:
time delay: max delay 300ms
reliability: 100% reliable
addCircle:
time delay: max delay 300ms
reliability: 100% reliable
moveOrResizeObject:
time delay: max delay 100ms
reliability: 100% reliable when complete; 90% reliable during drag
modifyText:
time delay: max delay 300ms
reliability: 100% reliable when complete; 95% reliable while typing

7.1.3 Gathering Group Event Data

Group event data was gathered by running user trials with realistic tasks and users. Two user tasks were defined for the case study using Pinelle and Gutwin's group task model [14]. The scenarios are described, as are the group types that would perform the tasks, but details about the tasks and subtasks are not shown here.

Scenario #1: Flowcharting the process of changing a tire.

Task description: The challenge is to make a complete flowchart of the process for changing a flat car tire. Each of the activities and decisions should be modeled appropriately with a consistent convention defined by the group. All flowchart components should be clearly labeled. Connectors between the flowchart components should be placed appropriately. Notes should be added to the flowchart where clarification is required. A legend should also be present of the flowchart. The end goal is to produce a flowchart that is complete and easy to comprehend.

Two separate groups of users were modeled to perform this task. Group #1 consisted of two experienced groupware users who had never previously performed the task. Group #2 consisted of four intermediate groupware users who had also not previously performed the task.

Scenario #2: Design your dream lab.

Task description: Designing the perfect HCI lab is a group effort since it must accommodate all of the members of the HCI lab. This task is to draw a 2D bird's eye view of an ideal lab together. The group should all include their input into the design but should also agree on what is produced in the end. Each of the components should be clearly drawn and easy to understand. The end product is a diagram of a perfect lab that all group members agree on.

One group type was modeled to perform this task. The group consisted of a mix of four groupware users with intermediate and advanced backgrounds. None of the users had previous experience performing this task.

Trials were run for each task with groups that fit the group model. The tasks were described to each group, but the method for performing the task was not discussed prior to the experiment. Scenario #1 was performed once by each group type defined.

Scenario #2 was performed twice, using a different group for each trial.

In total, 112,849 group events were gathered during the trials. These events were logged in files local to the machine that the events occurred on, so they were isolated by user. The timestamps on each of the log entries were determined by the local computer, so no common time was used between all users. The log entries were formatted in the same way across each of the trials. The format of the payload was maintained as it was generated by the system so that group event payload information would be correct. An example of the data gathered is shown in table 3.

Table 3: An example of group event data

time:802934225 move_telepointer(x:871.0 y:256.0)
time:802934266 move_telepointer(x:872.0 y:256.0)
time:802934583 (objorton_usask_ca>>>9357+6218.coords 829 253 889 317)
time:802934595 move_telepointer(x:871.0 y:256.0)
time:802934607 (objorton_usask_ca>>>9357+6218.coords 828 253 888 317)

7.1.4 Analyzing Data

The data analysis was done in two stages. The first stage served to extract a wide range of statistical information to produce a high level breadth-oriented view of the characteristics of GroupDraw. The data was analyzed in whole, separately for each group member, and separately for each group event type. The second stage used a more exploratory approach that followed up on any interesting hints with more detailed work to produce visualizations and draw out more specific statistics.

The analysis of the data was done using two tools, Ultraedit 9.00c and Microsoft Excel 2002. Ultraedit, a versatile text editor, was used to perform a variety of data reformatting tasks so that the data could be analyzed using Excel. The data had to be reformatted differently to support individual analysis tasks. For example, tasks that evaluated event payload had to preserve the original payload format, which was not usable for tasks that required more specific formats like counting occurrences of event types.

7.2 Characteristics Extracted

The analysis of the group event data identified many statistics and patterns in GroupDraw, some of which were general and some specific to a group or task. What follows is a sampling of some of the findings, but the length of this paper prevents the complete list of findings from being described. These examples were chosen to demonstrate the breadth of the findings, including characteristics of individual event types, different types of groups, specific tasks, and general characteristics based on all data.

7.2.1 Modify Text Event Patterns

One example of a pattern that was noted in a single event type that was not typical in other types of events was for modifyText events. It was noted that the maximum payload of any of the events was found in modifyText events, as was the largest variance of any of the payload sizes. The payloads of other event types were otherwise almost constant and rather predictable by event type. However, the variance of the payload in the modifyText events suggested that it would be more difficult to predict.

To better understand the behaviour of modify text events, the payload of each event was modeled over time for one user, as seen in figure 1. This revealed that modifyText events come in concentrated bursts and the payloads for sequential events are similar, but widely varied over the duration of the burst. Further analysis showed that most of the time, modifyText events varied by only 1 byte from the previous modify text message. However, when there was a change in payload that was larger than one byte, it could be large. Upon reviewing the logs, it became clear that this phenomenon was due to events being generated each time a user typed a key, and the event sent the complete text of the message, which varied by one character each time. Occasional large variances were due to pasting a string of characters at once.

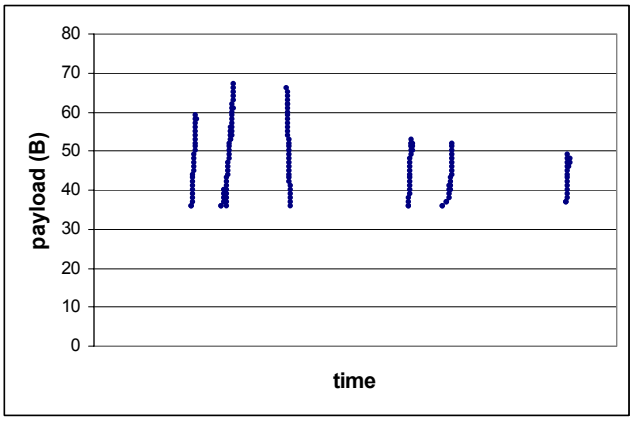

Figure 1: modifyText event payloads over time for one user.

7.2.2 Teamwork: Specialized Roles and Subgroups

The task to draw the process for changing a tire was recorded with two different types of groups: one with two users and the other with four users. The analysis of the data showed that the events generated by both members of the two user group shared very similar characteristics. Event frequencies were similar, event sizes and variances were very similar, and intervals between events were similar. Even the proportions of each event type were almost the same.

However, the events generated by the individuals in the group of four were dramatically different. The most active member generated 8.7 times more events than the least active member. Further analysis of the event types generated by each member suggested that the reason for the differences was that group members assumed specialized roles, while they did not in the two person group. The events generated by the activities associated with the roles accounted for these differences. Figure 2 shows the proportion of telepointer, modifyText, addBox, and addLine events for each of the four users. This breakdown suggests that users 2 and 3 were working as a team, with user 3 adding components, while user 2 annotated them. It also suggests that users 1 and 4 were working as a less specialized team, with user 1 drawing boxes and user 4 connecting the boxes with lines.

The significance of this finding is that specialized roles produce unique characteristics that allow them to be identified through group event data analysis. This also shows that users that are working together in subgroups can be identified by examining group event data.

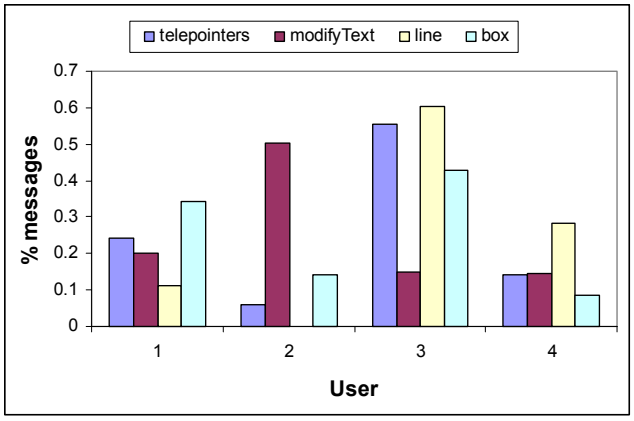

Figure 2: Breakdown of percentage of each event type generated by each user for the change tire task.

7.2.3 People Vary By Task

Both the tasks of flowcharting the change tire process and drawing the layout of the dream lab were performed by the same group of four. As mentioned above, users 2 and 3 worked in specialized roles to accomplish the change tire task. However, for the task of drawing the layout of the dream lab, all users produced more similar event breakdowns, suggesting that users 2 and 3 did not work together in the same way. However, user 3 still produced more events than the other three users, who each produced similar numbers of events and similar breakdowns of each event type, with some preferences being shown for the types of objects added.

This shows that groups adopt different collaborative strategies for different type of tasks. Additionally, it suggests that some users may be consistently more active than other users over a number of different tasks. More importantly, it further demonstrates that that information about how users behave can be acquired using automated methods, and it suggests that applications cannot necessarily expect similar behaviour from the same group for different tasks, but that it may be able to expect similar activity levels from individual group members. It also suggests that applications may be able to detect changes in tasks automatically based on changes in the activities of users. However, this is a very small sample size and it is important to note that these general observations are speculative and serve more to demonstrate how event characteristics can lead to hypotheses that may not otherwise be considered.

7.2.4 Consistent Event Characteristics

Some of the characteristics were observed to be fairly consistent for all users across every trial. This shows that some degree of generalization may be feasible, but more samples are required to explore this possibility more thoroughly.

The breakdown of events by type in GroupDraw across all groups and tasks showed that telepointer events accounted for about 83% of all of the events in the system. moveOrResizeObject events were also quite frequent, accounting for 12% of the total events, while 4% of the total events were modifyText events. The addLine, addCircle, addBox, and addText events (the 'add' events) together accounted for less than 1% of the total events generated. This event breakdown can be seen in figure 3.

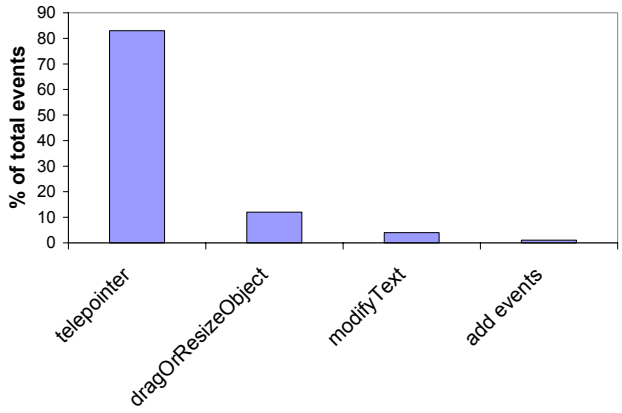

Figure 3: Frequency of each event type for all users and tasks.

The payloads of each of the events were quite similar, as the payload variance was low, with the exception being modifyText events, as mentioned above. moveOrResizeObject events produced about double the payload of telepointer events. 'add' events were in between the telepointer and moveOrResizeObject events in payload. The payloads of each event type can be seen in figure 4.

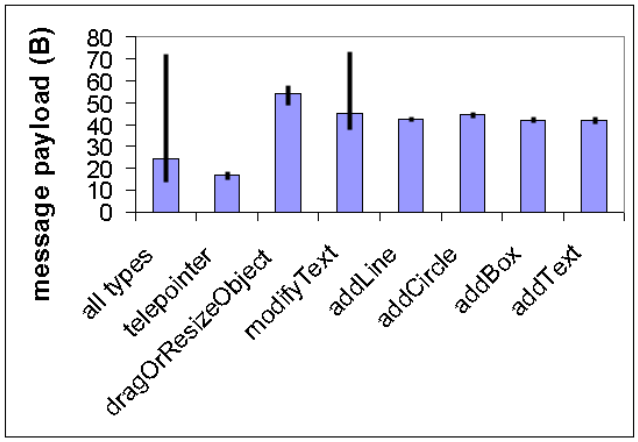

Figure 4: Payload averages and ranges for each event type.

The 'add' events, which were infrequent, were always separated by long time intervals, while the other types of events occurred in bursts. Telepointer events produced the longest bursts, followed by moveOrResizeObject events. The telepointer events and the moveOrResizeObject events had the same minimum interval times between events, and these event intervals were different for most users. The peak event frequencies occurred during dragOrResizeObject operations, where both telepointer events and dragOrResizeObject events were being generated simultaneously. modifyText events also occurred in bursts, although the intervals between events were much longer than the intervals of the dragOrResizeObject events.

Further analysis of the peak frequencies of the telepointer and dragOrResizeObject events revealed two main properties that were critical to performance of the system. The variance in peak frequency was found to be a result of the speed of the computers that ran the application. Faster computers would produce more events, while slow machines would produce fewer events. The

reason why some users experienced the same peak frequencies was that the machines they were using were of the same type. The other observation was that telepointer events are generated during moveOrResizeObject operations, resulting in the message frequency during these events being double the maximum telepointer frequency since both telepointer events and moveOrResizeObject events are being generated. Additionally, the information contained in the telepointer events information was found to be implied by the moveOrResizeObject event parameters. It was also noted that modifyText bursts did not take place at the same time as the telepointer or moveOrResizeObject bursts. The relationships between telepointer, moveOrResizeObject, and modifyText events can be seen in figure 5.

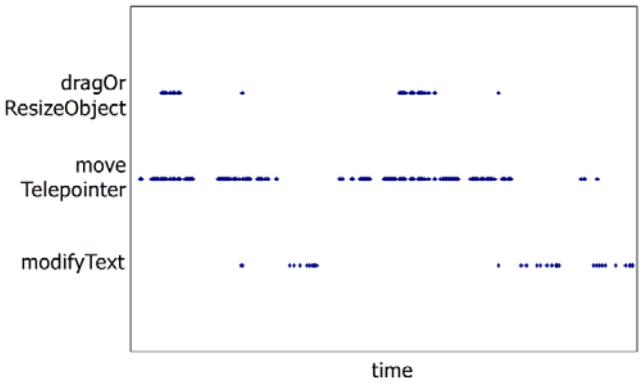

Figure 5: Scatterplot showing an example of the relationship between occurrences of event types over time.

7.3 Improving Performance

The characterization work revealed many opportunities for performance optimizations and also helped to identify where the most critical performance problems occured. To maintain consistency with the previous section, this section will describe only performance enhancements that relate to the characteristics noted above. However, more optimization ideas were noted as a result of the characterization work than are discussed here and many more would likely result from further analysis and consideration.

The frequencies and payloads of the event types generated by the system suggest an order of priority in which events should be considered for performance optimizations. The events that are the most critical to performance include telepointer, dragOrResizeObject, and modifyText events. 'add' events are very infrequent in comparison. Also, 'add' events do not occur in bursts and have predictable payloads, therefore 'add' events are unlikely to cause performance problems. The main concern with 'add' events is the reliability requirement of guaranteed delivery, which must be possible after other performance modifications are made.

dragOrResizeObject events are the most critical to performance since they can have frequencies that are as high as telepointer events do, and because their payloads are twice as large as those of telepointers. Additionally, the redundant telepointer messages that are sent during dragOrResizeObject operations must be addressed since this is the most critical performance bottleneck in the system. A simple suggestion that would improve the performance of GroupDraw dramatically is to stop sending telepointer messages whenever a dragOrResizeObject operation is underway and to use the data in the dragOrResizeObject events to imply the changes in telepointer positions. Another simple solution is to reduce the size of the payload using compression since its current format is very inefficient.

Both telepointer and dragOrResizeObject events are produced at rates determined by the speed of the computer the application runs on. This can result in very bad performance if there are several fast computers running the application and the network capabilities are low or there is another slow computer trying to participate that cannot process messages as quickly as the other machines generate them. A possible optimization is to control the rate at which these events are sent out by discarding extra events that exceed the QoS requirement for frequency. The QoS requirement for frequency of these event types was exceeded in each of the trials, so it can be assumed that the frequency of events will exceed the QoS requirements. Another possibility is to merge several events into a single message before sending to reduce the number of packets generated and to use a small buffer or traces at the receiver to play these events out. However, the client side play out technique must meet the QoS requirement for delay for these event types.

The QoS requirements for reliability allow for some of the telepointer and dragOrResizeObject events to be lost. Therefore, an efficient low-latency partially reliable delivery approach can be used to decrease overhead. Since these events occur in bursts, interleaving redundant information during bursts is a possibility. Also, it is possible to use error correction techniques that use redundancy during bursts because the payloads are small and the frequency during bursts is high. One constraint is that the last dragOrResizeObject event in a burst must be reliable so that the final location and size of the object are accurate. This can be achieved by detecting the end of the event burst using the statistical information extracted to identify when the end is and then send a final redundant, reliable message with the final position and size parameters of the object.

modifyText events are the next highest priority event type because they can occur in moderate frequency bursts and their payloads can be large and variable. The characteristic of modifyText operations that was the most significant was that all modifyText messages in a burst usually have a payload that is plus or minus one byte in difference from the previous event. It was also noted that the text parameter in the modifyText event only differed by one character or a continuous sequence of characters from the previous event. The modifyText events can be partially reliable as well, which suggests that each modifyText event should send a partially reliable message indicating the change in the text string rather than sending the entire string, which would make these messages very small be comparison since they would simply indicate the location of the change and the sequence of characters that would be added or removed. It was also noted that telepointer and dragOrResizeObject events are not generated during modifyText bursts, and that the peak frequencies of modifyText messages are much lower than the peaks of the application, so additional redundancy and periodic reliable messages can be sent during modifyText event bursts. This would both decrease the payload and maintain accuracy of the modifyText messages. Also, since event frequencies are known to be low, any feedback or control information generated

by the partially reliable techniques could be sent during modifyText event bursts with little effect on performance.

The possibility of modeling the statistical behaviour of certain roles played by group members can also lead to performance improvements by detecting the patterns that are characteristic of the roles and adapting to suit the role that has been assumed. For example, a group where one user has assumed a dedicated role of adding boxes and lines to a flowchart while another adds text to the flowchart entities could benefit from customized performance enhancements that suit their roles. One approach would be to loosen the QoS requirement for latency on telepointer messages since the users in this situation will be rarely reaching for the same item or duplicating work. This could allow several events for the person adding shapes to be combined into a single message and played out at the receiver using buffering. It could also allow the reliability of modifyText events to be increased for the other user since fewer events are generated for that user. Another possibility is to use a routing technique that would send some of the messages from the user adding shapes to the user adding text to distribute the messages to the rest of the group to reduce the total bandwidth required for distribution in a unicast environment.

8. DISCUSSION

The results of the case study suggest that extracting event characteristics from groupware can be an effective approach to identifying suitable existing performance improvements as well as to assist with developing new performance enhancing techniques. Some of the techniques that were identified could have been identified using other approaches, but this approach was both reasonably efficient and promoted thorough consideration and analysis. Other performance enhancing suggestions would have been difficult to identify without the extracted group event characteristics. The idea of statistically modeling the characteristics of certain roles and dynamically adapting the QoS requirements and performance techniques to better suit those roles is one that was inspired by the event characteristics and is an approach that we have never heard of before for groupware. The process of extracting characteristics and coupling them with performance enhancements was a valuable approach for considering performance enhancements for GroupDraw, and would likely be effective for other types of groupware applications as well.

The case study also demonstrated the importance of coupling QoS requirements with event characteristics when considering performance improvements. Although the QoS requirements defined in this example were quite simple, they allowed for much deeper analysis of what performance approaches might be possible and were consulted on numerous occasions for guidance.

The data gathered using this process will also have many additional merits to the overall understanding of how groupware works. If more people start using this type of approach, a rich and diverse set of group event logs will be produced. This data will have additional benefits to other areas such as simulating groupware, usability, user modeling, and task modeling. It may also be possible to generalize about some characteristics if convergent group event properties are extracted from a number of different data sets.

Future work will explore how event characteristics can be gathered automatically by groupware applications so that performance optimizations can be made adaptively based on the extracted characteristics of the group events. This is expected to lead to automatically customized performance enhancements that adapt to suit new user roles, interaction styles, and group tasks as they are discovered by the end users.

9. CONCLUSION

Extracting group event characteristics can lead to many possibilities of how the performance of groupware systems can be improved. Even though groupware is very difficult to characterize in general, group events can be characterized for specific tasks, implementations, and groups of users. A feasible, low cost approach to characterizing groupware includes defining the event types that the system produces; assigning QoS requirements to each event type; gathering data; and analyzing the data to extract group event characteristics. These characteristics can both help to identify existing techniques that can be applied to improve groupware performance as well as to promote new ideas for enhancing performance.

10. REFERENCES

[1] Balakrishnan, H., Padmanabhan, V., Seshan,S., Stemm, M., Katz, R. TCP behavior of a busy Internet server: Analysis and improvements. In Proc. INFOCOM, pages 252--262, March 1998.

[2] Bargh, M., ter Hofte, G. Analysis and simulation of the performance of real-time groupware systems: Definition of performance measures and indicators, performance study approach and simulation parameters. Enschede, Telematica Instituut, 2001.

[3] Bhola, S., Banavar, G., and Ahamad, M. Responsiveness and Consistency Tradeoffs in Interactive Groupware. In Proc. of the 7th ACM Conference on Computer Supported Cooperative Work, pages 79-88. 1998.

[4] Bolot, J-C. End-to-end packet delay and loss behaviour in the Internet. In Proc. SIGCOMM '93, pages 289-298, Sept. 1993.

[5] Engum, H., Iversen, J., Rein, O. Zereal: A semi-realistic simulator of Massively Multiplayer Online Games. Technical Report available at GameMining.net. 2002.

[6] Greenhalgh, C., Steve Benford, S., Craven, M. Patterns of network and user activity in an inhabited television event. Proc. Virtual reality software and technology. 1999.

[7] Gutwin, C. Effects of Network Delay on Group Work in Shared Workspaces. Proceedings of ECSCW 2001.

[8] Gutwin, C. Traces: Visualizing the Immediate Past to Improve Group Interaction. Proc. GI 2002.

[9] Isaacs, E., Walendowski, A., Whittaker, S., Schiano, D., Kamm, C. I M everywhere: The character, functions, and styles of instant messaging in the workplace. Proc. CSCW 2002.

[10] ISO/IEC 13236. Information Technology – Quality of Service: Framework. Available for purchase at http://www.iso.org. 1998.

[11] Lan, K., Heidemann, J. Rapid Model Parameterization from Traffic Measurements. USC Information Sciences Institute ISI-TR-561. 2002.

[12] Molnar, S., Maricza, I. (eds.) Source Characterization in Broadband Networks. Interim report, COST 257, Vilamoura, Portugal, January 1999. http://citeseer.nj.nec.com/article/molnar99source.html

[13] Paxson, V. Floyd, S. Why We Don't Know How to Simulate the Internet. In Proceedings of the 1997 Winter Simulation Conference, Atlanta GA, U.S.A., Dec. 1997.

[14] Pinelle, D., Gutwin, C. Group Task Analysis for Groupware Usability Evaluations. In submission to WetICE 2001.

[15] Roseman, M. and Greenberg, S. Building Real Time Groupware with GroupKit, A Groupware Toolkit. ACM Transactions on Computer Human Interaction, 3(1), p66-106. 1996.

[16] Rueda, A., Kinsner, W. A Survey of Traffic Characterization Techniques in Telecommunication Networks. Proc. IEEE Canadian Conference on Electrical and Computer Engineering. 1996.

[17] Sun, C., Chen, D. Consistency maintenance in real-time collaborative graphics editing systems. ACM Transactions on Computer-Human Interaction (TOCHI) March 2002. Volume 9 Issue 1.

[18] Stemm, M., Katz, R., Seshan, S. A Network Measurement Architecture for Adaptive Applications. Proceedings of IEEE Infocom 2000, March 2000.

[19] Vaghi, I., Greenhalgh, C., Benford, S. Coping with inconsistency due to network delays in collaborative virtual environments. Proceedings of the ACM symposium on Virtual reality software and technology December 1999.