The Design and Implementation of the Smartphone-based GroupNotes App for Ubiquitous and Collaborative Learning

Mark Reilly
School of Computer Science,
Engineering and Mathematics
Flinders University, Adelaide, Australia
m.reilly@flinders.edu.au

Haifeng Shen
School of Computer Science,
Engineering and Mathematics
Flinders University, Adelaide, Australia
haifeng.shen@flinders.edu.au

The lecture accounts for about half of staff-student interaction yet many students fail to engage in them as they find the environment is not conducive to the social learning paradigm they are accustomed to. Smartphones are rapidly replacing mobile phones within the student population and the screen size, connectivity and text entry speed provide a readily accessible platform for students to conduct ubiquitous and collaborative learning activities. In this paper, we present the design and implementation of a Smartphone App - $GroupNotes$, which allows a small group of students to participate in a real-time collaborative note-taking session using their own Smartphones in the lecture. Key technical contributions include an innovative collaborative note-taking interface, synchronisation of multi-user editable notes, and compression of operation buffers to reduce the consumption of network resources and Smartphone battery.

Smartphone, collaboration, synchronisation, compression, operational transformation, operational merging

1. INTRODUCTION

We proposed that a Smartphone-based approach to increasing student engagement in lectures is a viable and, most importantly, an accessible option for today's students (Reilly and Shen (2011)). A Smartphone-based collaborative note-taking application will provide a platform for small groups to create a self-motivating, social learning environment allowing them to collaborate silently in lectures while providing the necessary options and customisation to cater for the different learning styles of the individuals within those groups.

The choice of the Smartphone as the platform for a collaborative tool suitable for use within a lecture environment is due to the predicted ubiquity of such devices. At present the majority of university students either own or have access to a mobile phone (Litchfield et al. (2009)) and IDG predicted that by the start of 2010 sales of Smartphones within Australia would exceed 50% of new sales. While larger devices such as a $TabletPC$ may provide a more usable learning environment (Kam et al. (2005); Simon et al. (2008)), not all students are able, or willing to purchase the device, bring it to university with them, and then actually use it in the lecture environment.

The Smartphone is attractive to the student as there is no imposition associated with carrying an additional device, or in learning to use it or the new software since they have already learnt how to use the device voluntarily. There will be no additional monetary costs to use the device in the lecture environment as wireless access is already provided free therefore removing a major concern of students (Litchfield et al. (2009)).

Staff-student contact time at university is dominated by the lecture because it is recognised for its ability to uniformly deliver information not found in the standard texts, as well as the possibility to inspire and motivate students ranging from hundreds to perhaps thousands (Bligh, D. A. (2000)). Yet many students view some, or all of them as a waste of time (Hitchens and Lister (2009)) and either cease attending those lectures, or lose interest in as little as 20 minutes of a lecture due to no interaction with either the lecturer or other students in the vicinity (Bligh, D. A. (2000)).

The actual lecture environment itself is sub-optimal as it promotes the feeling that the students are at a performance. The shape of many lecture halls is that of an amphitheatre where the lecturer is far removed from the audience of students who are expected to sit in silence while the lecturing "performance" takes place. Students are disadvantaged in several ways with this environment, most notably because with the implied or actual prohibition on speaking, there is no opportunity to immediately interact with their fellow students, for clarification or validation of their understanding on the lecture content. Students feel isolated, part of an "anonymous mass" (Falkner and Munro (2009)). This change of teaching paradigm away from social learning as a member of an interactive (high school) classroom to that of a (mostly) passive audience member is seen as a cause of students failing to successfully complete their studies.

Note-taking during lectures is established as an important part of the process of codifying lecture content into knowledge that can be retrieved later. The current technology, in the form of soft keyboards such as $Swype$, $Swiftkey$ and $T9\ Trace$ allow text entry speed comparable or even faster than pen-and-paper handwriting. Our proposal is based on this observation (Reilly and Shen (2011)) and incorporates opportunities for the social learning attributes of challenging or validating one's current "knowledge" against others in a group in real time, which motivates a student to remain on task and to produce quality work. The application will also provide for the individuals who prefer to work alone, or who cannot keep up with multiple students during the lecture but would like to be part of a group with sharing of knowledge. In this instance, the student works alone, but stays motivated to produce quality work and to remain on task by the prospect of receiving multiple sets of notes from their other group members at the end of the lecture.

When the students are fully engaged, their attention span seems infinite, regardless of the teaching method (Matheson, C. (2008)). In this paper, we aim to design and implement such a Smartphone App - $GroupNotes$, a collaborative note-taking application that allows students to teach, correct, motivate and learn from their peers during the lectures. Key technical contributions include an innovative collaborative note-taking interface, synchronisation of multi-user editable notes, and compression of operation buffers to reduce the consumption of network resources and Smartphone battery.

The rest of the paper is organised as follows. The next section describes the application user interface. After that, we present the technical solutions for the note synchronisation and the operation buffer compression. Finally we conclude the paper with a summary of major contributions and future work.

2. APPLICATION USER INTERFACE

The Smartphone App user interface design is based on the results of a needs finding questionnaire given to third year computer science topic students at our university (Reilly and Shen (2011)). Working scenarios and screenshots are provided to illustrate particular design choices, with reference back to the educational requirements.

2.1. Device Characteristics

Today's Smartphone primarily uses a soft keyboard on a viewing screen ranging from approximately 3 to a maximum of 5 inches and our design focus will be on devices meeting these criteria.

2.2. Working Scenarios

The working scenarios provided show the application's user interface from the point of view of a single student's device.

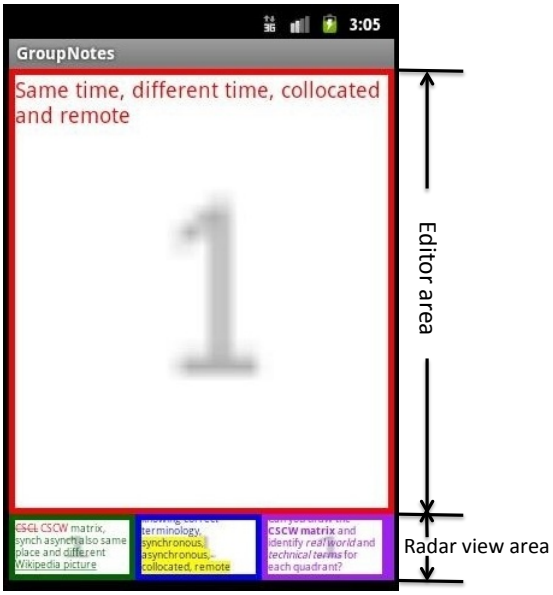


Figure 1: *1 viewable editor and 3 radar views*

Figure 1 shows the default screen view of the student after they have finished making a note. The screen is divided into the editor area (upper part) and the radar view area (lower part). The device owner can view and make notes in the editor area, while editors from their group members who have joined the session are shown in the radar view area. In this figure, $Student\ 1$'s editor is shown in the editor area on the screen, while the editors from $Students\ 2$, 3 and 4, from left to right at the bottom of the screen,

are shown in the radar view area. The background number in each editor, e.g., 1 in $Student\ 1$'s editor indicates that the student is making a note for the first lecture slide.

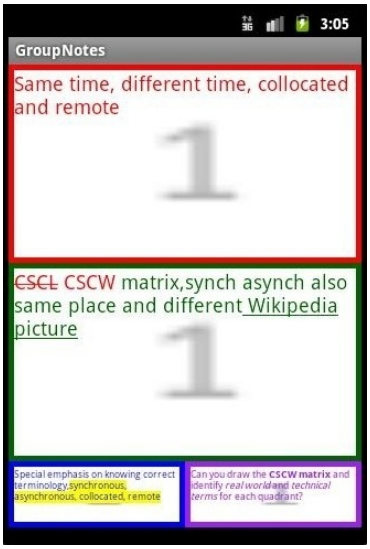


Figure 2: *2 viewable editors and 2 radar views*

In order for the student to view more editors, they need to drag one or more editors from the radar view area to the editor area. In this example, there are four individual editors available representing the four students who are part of this group. The group may be made up of more or less students; four was identified from the questionnaire as being the maximum number of concurrent users an individual could keep up with cognitively during a lecture. The available screen real estate also limits the number of viewable editors possible.

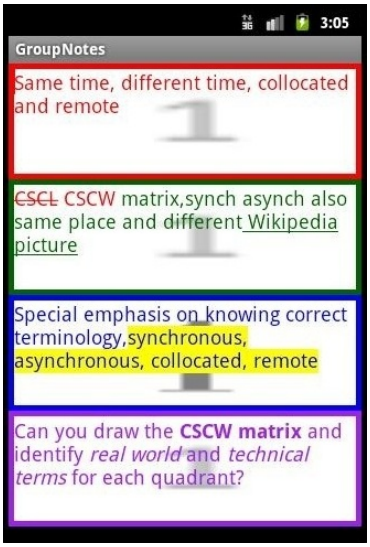


Figure 3: *All 4 viewable editors*

Figure 2 shows that the student has chosen to view two editors at the same time. This occurs by dragging $Student\ 2$'s editor radar view (the bottom left editor in Figure 1) up to the editor area. To return to a single editor, which may or may not be the device owner's editor, would require the reverse action, i.e. dragging the unwanted editor down to the radar view area. Figure 3 shows all four editors that take up all available screen real estate. It is worth pointing out that while a student can view up to four editors at the same time, when they start doing text entry in an editor, that editor will occupy the entire editor area, pushing the rest of editors from the editor area to the radar view area, primarily due to the significant screen real estate taken by the soft keyboard.

2.3. Text Editing Functions

At present the text editor only has basic functions including the ability to change the colour of the text (system enforces the text colour and users are not allowed to), present text in plain, bold or italic, highlight the background of selected text, strikethrough selected text or underline selected text. These features will be accessed through a floating toolbar which will appear whenever text is selected. Figure 3 illustrates all of these features. $Student\ 1$ is using plain text in red in their own editor. $Student\ 2$ has written notes in green in their editor to which $Student\ 1$ has performed a strikethrough of the acronym CSCL and then entered the correct acronym CSCW; both changes are in a different colour to identify an input from other than the owner of the editor. $Student\ 2$ has also highlighted something they consider important in their notes and has underlined the words Wikipedia picture. $Student\ 3$ is using blue text and has provided emphasis on four words from their notes through highlighting. $Student\ 4$ using the colour purple has used bold on the words CSCW matrix and italicised real world and technical terms for emphasis.

2.4. User Identification

The questionnaire strongly suggested that users should be differentiated on the individual's device through the use of a unique colour for each group member. This colour borders the editor where that user writes their notes and is illustrated in each of Figures 1-3. The colour also identifies specific user generated content which is entered into another group members writing area, such as Figure 2 where $Student\ 1$ (in red) performed a strikethrough on the text CSCL written by the owner of editor 2 (in green). CSCW is the new text written in this green area, also in red. This allows other group members to determine who has made the change.

3. NOTE SYNCHRONISATION

In a real-time collaborative note-taking session consisting of up to 4 students, each student is allocated a dedicated multi-page note that corresponds to the the number of lecture slides. Each note can be jointly edited by the 4 students; therefore, the note-taking session is actually composed of 4 parallel collaborative editing sessions, one for each note. We devised an optimistic concurrency control solution to synchronise each note by leveraging the $GroupNotes\ Server$ as the mediator. In this solution, each note is replicated in every student's App and operations performed in one App are broadcast to all other Apps.

3.1. Operational Transformation

There are two types of operations involved in editing a note: $Ins/Del[\langle page, position\rangle, length, text]$ denotes inserting/deleting a piece of $text$ of $length$ at the $position$ in the $page$ of the note. Updating an attribute of a piece of text, e.g., highlighting the text, is represented by deletion of the text with the old attribute value.

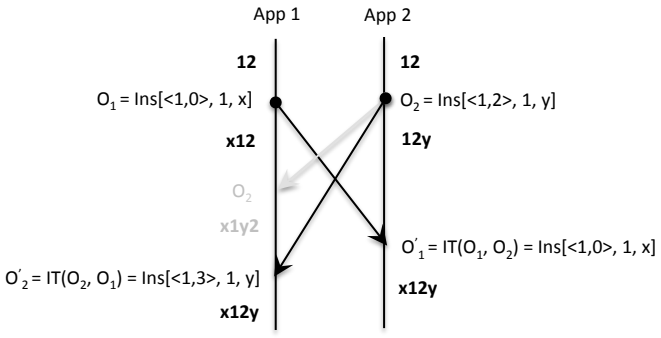


Figure 4: *Operational transformation*

As shown in Figure 4, when operation O_2 = $Ins[\langle 1, 2\rangle, 1, y]$ generated at $App\ 2$ arrives at $App\ 1$, it cannot be replayed as-is because the concurrent operation O_1 = $Ins[\langle 1, 0\rangle, 1, x]$ has changed the context in which O_2 was defined, i.e., "12", to "x12". We adopt the *operational transformation* technique (Sun and Ellis (1998)) to transform O_2 against O_1 in such a way that O_2' = **IT**(O_2, O_1) = $Ins[\langle 1, 3\rangle, 1, y]$ has effectively included the impact of the concurrent operation O_1 (Sun et al. (1998)).

3.2. Note Synchronisation

The note synchronisation solution consists of an operation broadcast protocol, an operation replaying algorithm, and a set of collaborative editing session management protocols.As shown in Figure 5, the server runs m ($m \geq 1$) collaborative note-taking sessions; each session has up to 4 collaboratively edited notes. There are n ($n \geq 1$) Apps connected to the server and up to 4 Apps, e.g., App i, j, k, and l ($1 \leq i, j, k, l \leq n$), can share the same session, e.g., session p ($1 \leq p \leq m$). For each note, the server maintains a master note MN, a master incoming operation buffer MIB storing all operations that should be executed on MN to get its latest state, and a server-side incoming operation buffer for each App involved in the same note, e.g., SIB_i, SIB_j, SIB_k, and SIB_l in session p, storing remote operations that have been received by the server but are yet to be received by the corresponding App.

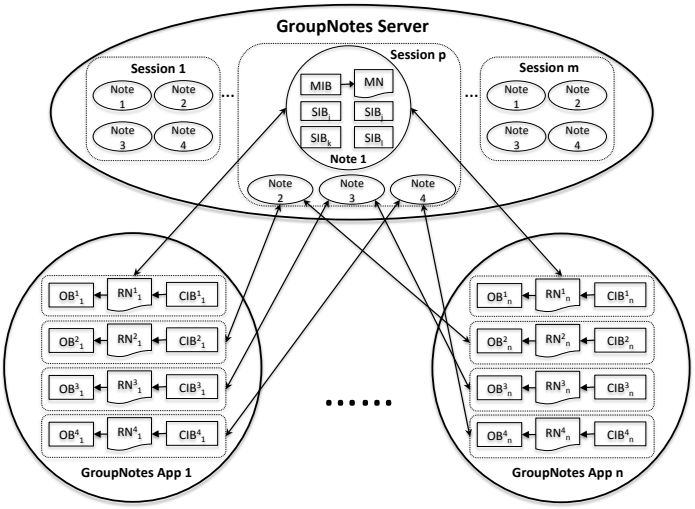


Figure 5: *Note synchronisation*

Each App can write into any of the 4 notes at any time, therefore it needs to separately synchronise these 4 notes. For each note, the App maintains a replica of the note RN, an outgoing operation buffer OB storing locally generated operations on RN, and a client-side incoming operation buffer CIB storing remote operations that have already been received by the App and will be replayed on RN. In Figure 5, RN_j^i, OB_j^i, and CIB_j^i ($1 \leq i \leq 4$ and $1 \leq j \leq n$) are App j's replica, OB, and CIB for note i respectively.

3.3. Operation Broadcast

A note synchronisation process consists of two sub-processes: broadcast local operations and replay remote operations. Suppose App k ($1 \leq k \leq n$) is involved in note r ($1 \leq r \leq 4$) in the note-taking session p ($1 \leq p \leq m$), the following protocol is executed by the App to broadcast local operations generated on note replica RN_k^r from OB_k^r.

Protocol 1 Operation Broadcast Protocol

1. At App k, move all available operations in OB_k^r to the operation sequence $\overrightarrow{OB_k^r}$. If CIB_k^r is

not empty, invoke the procedure **SLOT**($\overrightarrow{OB}_k^r$, CIB_k^r) to transform the operation sequence $\overrightarrow{OB}_k^r$ with the operation sequence in CIB_k^r.

2. App k establishes a network connection with the server and then sends a request $\langle$*BROADCAST*, k, p, r, $\overrightarrow{OB}_k^r\rangle$ to the server.
3. When the server receives the request from App k, it performs an atomic synchronisation process on note r in session p:
   (a) lock note r, including the *MIB* and all the SIB_i ($1 \leq i \leq n$);
   (b) if SIB_k is not empty, invoke the procedure **SLOT**($\overrightarrow{OB}_k^r$, SIB_k) to transform $\overrightarrow{OB}_k^r$ with the operation sequence in SIB_k;
   (c) append $\overrightarrow{OB}_k^r$ to *MIB* and all SIB_j ($1 \leq j \leq n$, $j \neq k$);
   (d) if SIB_k is not empty, move all operations in SIB_k to the operation sequence $\overleftarrow{SIB}_k$ and send a response $\langle$*BROADCAST*, $\overleftarrow{SIB}_k\rangle$ to App k, otherwise, send a dummy response $\langle$*BROADCAST*$\rangle$ to App k; and
   (e) unlock note r in session p.
4. When App k receives the response from the server, if the response is not dummy, it appends the piggyback operation sequence $\overleftarrow{SIB}_k$ to CIB_k^r.

The **SLOT** transformation control algorithm, which symmetrically transforms two context-equivalent sequences Sq_a = [$O_{a,0}$ $\cdots$ $O_{a,m-1}$] (m = $|Sq_a|$) and Sq_b = [$O_{b,0}$ $\cdots$ $O_{a,n-1}$] (n = $|Sq_b|$), and returns transformed ones Sq_a^b and Sq_b^a, can be found in an earlier work (Shen and Sun (2002)).

3.4. Operation Replay

The other sub-process involved in a note synchronisation is to replay remote operations stored in an App's CIB. The above operation replay algorithm executes remote operations in CIB_k^r to complete the synchronisation process on note r at App k.

It is worth pointing out that because local and remote operations need to modify the same replica of note r at App k, execution of local operations and replaying of remote operations must be mutually exclusive. In case of contention between local and remote operations, local operations must be given the priority to ensure good local response. Otherwise, if all remote operations in CIB_k^r were replayed as a continual stream, local operations would suffer starvation, resulting in poor local response. To minimise the impact on the local response, each remote operation in CIB_k^r should "give way" to new local operations before being replayed.

ALGORITHM 1: Operation Replay Algorithm (CIB_k^r, OB_k^r)

```
1:  if (|CIB_k^r|== 0) then
2:    return
3:  end if
4:  start ← 0
5:  if (|OB_k^r|> start) then
6:    e ←|OB_k^r| −1
7:    SLOT(CIB_k^r, OB_k^r[start, end])
8:    start ← end + 1
9:  end if
10: repeat
11:   give_way()
12:   lock_replica()
13:   if (|OB_k^r|> start) then
14:     end ←|OB_k^r| −1
15:     SLOT(CIB_k^r, OB_k^r[start, end])
16:     start ← end + 1
17:   end if
18:   O ← CIB_k^r[0]
19:   execute(O)
20:   CIB_k^r.remove(0)
21:   unlock_replica()
22: until (|CIB_k^r|== 0 or HOB's turn)
```

Due to the space space limitation, collaborative editing session management protocols are not given in the paper.

4. OPERATION BUFFER COMPRESSION

The network connection between an App and the server is based on Wi-Fi, where the network resource is always limited. Frequent broadcast and replay of every individual operation is undesirable as it will consume too much network resource as well as the battery of the Smartphone running the App. It is also unnecessary as students are only interested in what note their peers are taking rather than micro-step operations. Therefore, we devised an operation buffer compression technique to compress each App's OB so that only the net effects of the operations in each OB will be broadcast and replayed. The compression technique can significantly reduce both the size of an OB and the number of operations in it.

4.1. Operation Relationships

Definition 1 *(Operation Context)*

Given an operation O, its context, denoted by Υ_O, is the state of the note on which O is defined.

Definition 2 *(Operation Context Preceding Relation)*

Given two operations O_a and O_b, O_a is context-preceding O_b, or O_b is context-succeeding O_a, denoted by $O_a \mapsto O_b$, iff $\Upsilon_{O_b} = \Upsilon_{O_a} \vdash O_a$, where '$\vdash$' is the operation execution operator.

Υ_{O_a} is the context (i.e., state of the note) on which O_a is defined. After the execution of O_a on Υ_{O_a}, the new context can be described by $\Upsilon_{O_a} \vdash O_a$. If O_b is defined on this new context, i.e., $\Upsilon_{O_b} = \Upsilon_{O_a} \vdash O_a$, then O_b is context-succeeding O_a. In general, if the initial context of a note is Φ and n operations $O_1, \cdots, O_n$ have been executed in sequence, i.e., $OB = [O_1, \cdots, O_n]$, then the final context can be described by $\Phi \vdash O_1 \vdash \cdots \vdash O_n$. For $\forall i \in \{2, \cdots, n\}$, $O_{i-1} \mapsto O_i$ because $\Upsilon_{O_1} = \Phi$ and $\Upsilon_{O_i} = \Upsilon_{O_{i-1}} \vdash O_{i-1}$.

Because operations performed in different pages of a note cannot be compressed, for the good of presentation, operation O is represented as $Ins/Del[position, length, text]$, where **P**($O$) = $position$ denotes O's position parameter, **N**(O) = $length$ denotes O's length parameter, **S**(O) = $text$ denotes O's text parameter, and **T**(O) = Ins/Del denotes O's type parameter. The following definitions are used to describe the relationships between any two operations in an OB.

Definition 3 *Operation overlapping relation "$\oplus$"*

Given two operations O_a and O_b where $O_a \mapsto O_b$, O_a and O_b are *overlapping*, denoted as $O_a \oplus O_b$, **iff** (if and only if) one of following conditions holds:

1. **T**(O_a) = **T**(O_b) = *Ins*, and **P**(O_a) < **P**(O_b) < **P**(O_a)+**N**(O_a).
2. **T**(O_a) = **T**(O_b) = *Del*, and **P**(O_b) < **P**(O_a) < **P**(O_b)+**N**(O_b).
3. **T**(O_a) = *Ins* and **T**(O_b) = *Del*, and **P**(O_a) ≤ **P**(O_b) < **P**(O_a)+**N**(O_a) or **P**(O_b) ≤ **P**(O_a) < **P**(O_b)+**N**(O_b).

4.2. Operational Merging

Two operations O_a and O_b are overlapping if their effect regions are overlapping. First, if an insertion operation O_b inserts a string that falls into the effect region of a previous insertion operation O_a, then O_a and O_b are overlapping. Second, if a deletion operation O_a deletes a range that falls into the effect region of a later deletion operation O_b, then O_a and O_b are overlapping. Third, if an insertion operation O_a inserts a string which or part of which falls into the effect region of a later deletion operation O_b, then O_a and O_b are overlapping. Finally, under no circumstance could an insertion operation overlap with a previous deletion operation because there is no way for a string to be inserted into a nonexistent string (i.e., a string that has already been deleted).

Definition 4 *Operation adjacent relation "$\ominus$"*

Given two operations O_a and O_b where $O_a \mapsto O_b$, O_a and O_b are *adjacent*, denoted as $O_a \ominus O_b$, **iff** one of the following conditions holds:

1. **T**(O_a) = **T**(O_b) = *Ins*, and **P**(O_b) = **P**(O_a) or **P**(O_b) = **P**(O_a)+**N**(O_a).
2. **T**(O_a) = **T**(O_b) = *Del*, and **P**(O_a) = **P**(O_b)+**N**(O_b) or **P**(O_a) = **P**(O_b).

The same type of two operations are adjacent if their effect regions are adjacent. If an insertion operation O_b inserts a string that is adjacent to the string inserted by a previous insertion operation O_a, then O_a and O_b are adjacent. If a deletion operation O_b deletes a range that is adjacent to the range deleted by a previous deletion operation O_a, then O_a and O_b are adjacent.

Definition 5 *Operation disjointed relation "$\odot$"*

Given two operations O_a and O_b, O_a and O_b are *disjointed*, denoted as $O_a \odot O_b$, **iff** neither $O_a \oplus O_b$ nor $O_a \ominus O_b$.

Two adjacent operations can be merged into one operation by concatenating their effect regions. In this way, the number of operations in an OB can be reduced by one. The same type of two overlapping operations can be merged into one operation by combining their effect regions. In this way, the number of operations in the OB can be reduced by one. Different types of two overlapping operations can be merged in such a way that the overlapping region is removed from both operations. In this way, the size of the OB can be reduced and the number of operations in the log could be reduced by one or two if the effect region of one operation totally falls into the effect region of the other operation or the effect regions of the two operations are completely overlapping. The following functions are defined to merge two operations in an OB.

ALGORITHM 2: OM(O_a, O_b): (O'_a, O'_b)

Require: $O_a \mapsto O_b$
Ensure: $O'_a \odot O'_b$
  if (**T**(O_a) == *Ins* **and T**(O_b) == *Ins* **then**
    return $OM\_II(O_a, O_b)$
  else if (**T**(O_a) == *Ins* **and T**(O_b) == *Del* **then**
    return $OM\_ID(O_a, O_b)$
  else if (**T**(O_a) == *Del* **and T**(O_b) == *Del* **then**
    return $OM\_DD(O_a, O_b)$
  else
    return (O_a, O_b)
  end if

ALGORITHM 3: OM_DD(O_a, O_b): (O'_a, O'_b)

if (**P**(O_a) ≥ **P**(O_b) **and** **P**(O_a) ≤ **P**(O_b)+**N**(O_b)) **then**
  head ← **substring**(**S**(O_b), 0, **P**(O_a)-**P**(O_b))
  tail ← **substring**(**S**(O_b), **P**(O_a)-**P**(O_b), **N**(O_b))
  T(O'_a) ← *Del*; **P**(O'_a) ← **P**(O_b)
  N(O'_a) ← **N**(O_a)+**N**(O_b)
  S(O'_a) ← *head*+**S**(O_a)+*tail*
  return (O'_a, **I**)
else
  return (O_a, O_b)
end if

ALGORITHM 4: OM_II(O_a, O_b): (O'_a, O'_b)

if (**P**(O_b) ≥ **P**(O_a) **and** **P**(O_b) ≤ **P**(O_a)+**N**(O_a)) **then**
  head ← **substring**(**S**(O_a), 0, **P**(O_b)-**P**(O_a))
  tail ← **substring**(**S**(O_a), **P**(O_b)-**P**(O_a), **N**(O_a))
  T(O'_a) ← *Ins*; **P**(O'_a) ← **P**(O_a)
  N(O'_a) ← **N**(O_a)+**N**(O_b)
  S(O'_a) ← *head*+**S**(O_b)+*tail*
  return (O'_a, **I**) {**I** is an identity (null) operation}
else
  return (O_a, O_b)
end if

OM_II(O_a, O_b) is defined to merge two insertion operations O_a and O_b where $O_a \mapsto O_b$. If $O_a \oplus O_b$ or $O_a \ominus O_b$, O_a and O_b will be merged into a single insertion operation O'_a that integrates the effect regions covered by both O_a and O_b. **OM_DD**(O_a, O_b) is defined to merge two deletion operations O_a and O_b where $O_a \mapsto O_b$. If $O_a \oplus O_b$ or $O_a \ominus O_b$, O_a and O_b will be merged into a single deletion operation O'_a that integrates the effect regions covered by both O_a and O_b. **OM_ID**(O_a, O_b) function is defined to merge an insertion operation O_a and a deletion operation O_b where $O_a \mapsto O_b$. If $O_a \oplus O_b$, the overlapping region will be removed from both O_a and O_b in such a way that the common substring *ST* inserted by O_a but later deleted by O_b is eliminated from both O_a and O_b. **OM_ID**(O_a, O_b) is the most effective merging function that can dramatically reduce the size and the number of operations in an OB by removing redundant operations or redundant information in operations. Due to the space limitation, the **OM_ID** function is not given in the paper.

4.3. The Compression Algorithm

Definition 6 *Maximally compressed OB "Ω_{OB}"*

Given an OB, Ω_{OB} denotes its *maximally compressed* form in which for $\forall O_i, O_j \in OB$ ($0 \le i, j < |OB|$ and $i \ne j$), $O_i \odot O_j$.

We devised a compression algorithm based on operational merging, which can achieve maximal compression. First, the algorithm exhaustively examines every two operations in an OB, including those that are not physically located one after another in the buffer. In the aforesaid example, the compressed OB is [O_1^3, O_3^3, O_5], where $O_1^3 \odot O_3^3$ and $O_3^3 \odot O_5$, but the relationship between O_1^3 and O_5 is not examined. We adopt the operational transformation technique to reshuffle an OB through the following **LTranspose** (OB, i, j) ($0 \le i < j < |OB|$) function, which contextually swaps operations $OB[i]$ and $OB[j]$.

ALGORITHM 5: LTranspose(OB, i, j)

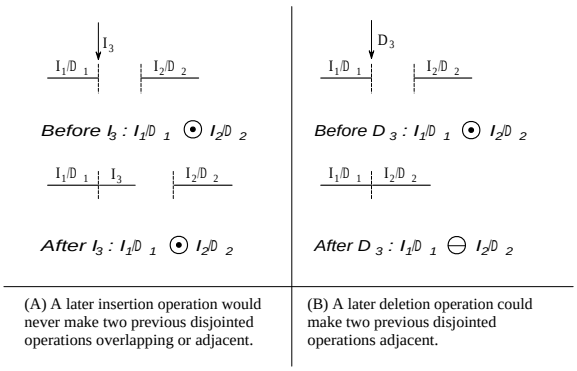
for ($k \leftarrow j; k > i; k \leftarrow k-1$) **do**
  $O_{k-1} \leftarrow OB[k-1]$; $O_k \leftarrow OB[k]$
  Transpose(O_{k-1}, O_k)
  $OB[k-1] \leftarrow O_k$
  $OB[k] \leftarrow O_{k-1}$
end for

ALGORITHM 6: Transpose(O_a, O_b): (O'_b, O'_a)

Require: $O_a \mapsto O_b$
Ensure: $O'_b \mapsto O'_a$
  $O'_b \leftarrow$ **IT** (O_b, $\overline{O_a}$) {$\overline{O_a}$ is O_a's inverse}
  $O'_a \leftarrow$ **IT** (O_a, O'_b)

  return (O'_b, O'_a)

Applying **LTranspose** (OB, 1, 2) to the aforesaid example would result in a reshuffled OB = [O_3^4, O_1^4, O_5], where $O_3^4 = Del[1, 2, yz]$, $O_1^4 = Ins[1, 1, c]$, and $O_3^4 \mapsto O_1^4 \mapsto O_5$. It is clear that $O_1^4 \ominus O_5$ and the OB can be further compressed to OB = [O_3^4, O_1^5], where $O_1^5 = Ins[1, 4, c123]$. This OB has achieved maximal compression, i.e., $OB = \Omega_{OB}$, achieving 60% reduction in operation number and 57% reduction in buffer size.



(A) A later insertion operation would never make two previous disjointed operations overlapping or adjacent.

(B) A later deletion operation could make two previous disjointed operations adjacent.

Figure 6: *Hidden adjacent operations*

Second, the algorithm can discover hidden adjacent operations, i.e., these operations were initially disjointed and then made adjacent by subsequent operations. As shown in Figure 6(A), operations I_2 (an insertion operation) or D_2 (a deletion operation) and I_1/D_1 were initially disjointed. If a new insertion operation I_3 inserts a sequence of characters, for instance between the effect regions covered by I_1/D_1 and I_2/D_2, it can never make I_2/D_2 and I_1/D_1 adjacent. In contrast, a later deletion operation could create new adjacent relations among previous disjointed operations. As shown in Figure 6(B), operations I_2/D_2 and I_1/D_1 were initially disjointed. If a new deletion operation D_3 deletes all characters between the effect regions covered by I_1/D_1 and I_2/D_2, I_2/D_2 and I_1/D_1 then becomes adjacent.

To address this issue, the algorithm transposes all deletion operations to the left side of all insertion operations in an OB in order to effectively detect the hidden adjacent relations created by these deletion operations. Therefore, the algorithm first merges all deletion operations with other operations and then merges insertion operations with the rest of the operations. In this way, insertion operations must have already taken into account the effects of all deletion operations and consequently the algorithm can merge hidden adjacent operations. As a result, a maximally compressed OB must look like [D_1, $\cdots$, D_r, I_1, $\cdots$, I_s] where D_i ($1 \leq i \leq r$) is a deletion operation and I_j ($1 \leq j \leq s$) is an insertion operation.

The **COMET** (Compression by Operational Merging & Transformation) compression algorithm can achieve maximal compression, as described by the following theorem. Given an OB = [O_1, $\cdots$, O_n] storing a list of user-issued operations on a note, **COMET**(OB) = Ω_{OB} = [EO_1, $\cdots$, EO_m] ($m \leq n$), where EO_i (1 $\leq i \leq m$) is referred to as an *effective operation*. A list of effective operations is the minimal list of essential operations in transforming a note from its initial state to the final state while preserving the intentions of user-issued editing operations.

ALGORITHM 7: COMET(OB): ($\widetilde{OB}$)

$\widetilde{OB}^d \leftarrow$ **COMEType**(OB, *Del*)
$\widetilde{OB}^i \leftarrow$ **COMEType**(OB, *Ins*)

return $\widetilde{OB} \leftarrow \widetilde{OB}^d + \widetilde{OB}^i$

Theorem 1 *Given any OB, if* **COMET***(OB) = $\widetilde{OB}$, then it must be $\widetilde{OB} = \Omega_{OB}$.*

Proof: For |OB| = 1, $\Omega_{OB} = OB$ and $\widetilde{OB} = OB$. The theorem holds.

ALGORITHM 8: COMEType(OB, Type): ($\widetilde{OB}^x$)

while $((i \leftarrow \mathbf{lastOp}(OB, Type)) \geq 0$ **and** $merged ==$ **false**) **do**
  if ($i == 0$) **then**
    $\widetilde{OB}^x$.**add**($OB[i]$)
    OB.**remove**(i)
    $merged \leftarrow$ **true**
  else
    for ($j \leftarrow i-1$; $j > 0$; $j \leftarrow j-1$) **do**
      (O_i, O_j) $\leftarrow$ **OM**($OB[j]$, $OB[i]$)
      if (O_i == O_j == **I**) **then**
        OB.**remove**(i)
        OB.**remove**(j)
        $merged \leftarrow$ **true**
      else if (O_j == **I**) **then**
        $OB[i] \leftarrow O_i$
        OB.**remove**(j)
        $i \leftarrow i-1$
      else if (O_i == **I**) **then**
        $OB[j] \leftarrow O_j$
        OB.**remove**(i)
        $merged \leftarrow$ **true**
      else
        $OB[j] \leftarrow O_j$
        $OB[i] \leftarrow O_i$
        LTranspose(OB, j, i)
        $i \leftarrow i-1$
      end if
    end for
    if ($i == 0$) **then**
      $\widetilde{OB}^x$.**add**($OB[i]$)
      OB.**remove**(i)
    end if
  end if
end while

ALGORITHM 9: lastOp(OB, Type): i ($0 \leq i < |OB|$)

$i \leftarrow |OB| - 1$; $found \leftarrow$ **false**
while ($i \geq 0$ **and** $found ==$ **false**) **do**
  if (**T**($OB[i]$) == *Type*) **then**
    $found \leftarrow$ **true**
  else
    $i \leftarrow i-1$
  end if
end while

return i

For |OB| = n, hypothesize that the theorem holds, that is, given OB = [$O_1, \cdots, O_n$], if $\widetilde{OB}$ = [$D_1, \cdots D_r$, $I_1, \cdots, I_s$], where all D_i (1 $\leq i \leq r$) are disjointed deletion operations, all I_j (1 $\leq j \leq s$) are disjointed insertion operations, and $D_i \odot I_j$.

For |OB| = $n+1$, OB = [$O_1, \cdots, O_n, O_{n+1}$], $\widetilde{OB}$ = **COMET**(OB) = **COMET**([$O_1, \cdots, O_n, O_{n+1}$]) = **COMET**([[$O_1, \cdots, O_n$], [$O_{n+1}$]]) = **COMET**([**COMET**([$O_1, \cdots, O_n$]), **COMET**([O_{n+1}])]) = **COMET**([[$D_1, \cdots, D_r, I_1, \cdots, I_s$], [$O_{n+1}$]]) = **COMET**([$D_1, \cdots, D_r, I_1, \cdots, I_s, O_{n+1}$]).

If O_{n+1} is an insertion operation, then for $\forall D_i$ (1 $\leq i \leq r$), $D_i \odot O_{n+1}$. Therefore $\widetilde{OB}$ = **COMET**([$D_1, \cdots, D_r, I_1, \cdots, I_s, O_{n+1}$]) = [$D_1, \cdots, D_r$, **COMET**([$I_1, \cdots, I_s, O_{n+1}$])].

1. Given $\forall I_j$ (1 $\leq j \leq s$), if $I_j \odot O_{n+1}$, then $\widetilde{OB}$ = **COMET**([$D_1, \cdots, D_r, I_1, \cdots, I_s, O_{n+1}$]) = [$D_1, \cdots, D_r, I_1, \cdots, I_s, O_{n+1}$], where all operations are disjointed. That is $\widetilde{OB} = \Omega_{OB}$ and the theorem holds.

2. If $\exists I_k$ (1 $\leq k \leq s$), where $I_k \oplus/\ominus O_{n+1}$, O_{n+1} will be merged into I_k. **COMET**([$I_1, \cdots, I_k, \cdots, I_s, O_{n+1}$]) = [$I_1, \cdots, I'_k, \cdots, I_s$], where all operations are disjointed. So $\widetilde{OB}$ = **COMET**([$D_1, \cdots, D_r, I_1, \cdots, I_s, O_{n+1}$]) = [$D_1, \cdots, D_r, I_1, \cdots, I'_k, \cdots, I_s$], where all operations are disjointed. That is $\widetilde{OB} = \Omega_{OB}$ and the theorem holds.

If O_{n+1} is a deletion operation, the proof is as follows.

1. Given $\forall D_i$ (1 $\leq i \leq r$) or $\forall I_j$ (1 $\leq j \leq s$), if $D_i \odot O_{n+1}$ and $I_j \odot O_{n+1}$, then $\widetilde{OB}$ = **COMET**([$D_1, \cdots, D_r, I_1, \cdots, I_s, O_{n+1}$]) = **COMET**([$D_1, \cdots, D_r, O'_{n+1}, I'_1, \cdots, I'_s$]) = [**COMET**([$D_1, \cdots, D_r, O'_{n+1}$]), **COMET**([$I'_1, \cdots, I'_s$])]. The deletion operation O_{n+1} would never create new adjacent relations among [$D_1, \cdots, D_r$], so **COMET**([$D_1, \cdots, D_r, O'_{n+1}$]) = [$D_1, \cdots, D_r, O'_{n+1}$], where all deletion operations are disjointed. However, the deletion operation O_{n+1} may create a new adjacent relation between I_l and I_q (1 $\leq l, q \leq s$). As a result, **COMET**([$I'_1, \cdots, I'_s$]) = [$I'_1, \cdots, I^{"}_l, \cdots, I'_{q-1}, I'_{q+1}, \cdots, I'_s$], where all insertion operations are disjointed. So $\widetilde{OB}$ = **COMET**([$D_1, \cdots, D_r, I_1, \cdots, I_s, O_{n+1}$]) = [$D_1, \cdots, D_r, O'_{n+1}, I'_1, \cdots, I'_s$] or [$D_1, \cdots, D_r, O'_{n+1}, I'_1, \cdots, I^{"}_l, \cdots, I'_{q-1}, I'_{q+1}, \cdots, I'_s$], where all operations are disjointed. That is $\widetilde{OB} = \Omega_{OB}$ and the theorem holds.

2. Given $\forall I_j$ (1 $\leq j \leq s$), where $I_j \odot O_{n+1}$, but $\exists D_k$ (1 $\leq k \leq r$), where $D_k \oplus/\ominus O_{n+1}$, then O_{n+1} will be merged into D_k. So $\widetilde{OB}$ = **COMET**([$D_1, \cdots, D_r, I_1, \cdots, I_s, O_{n+1}$]) = [**COMET**([$D_1, \cdots, D_r, O'_{n+1}$]), **COMET**([$I'_1, \cdots, I'_s$])]. **COMET**([$D_1, \cdots, D_r, O'_{n+1}$]) = [$D_1, \cdots, D'_k, \cdots, D'_r$], where all deletion operations are disjointed. O_{n+1} may create a new adjacent relation between I_l and I_q (1 $\leq l, q \leq s$). As a result, **COMET**([$I'_1, \cdots, I'_s$]) = [$I'_1, \cdots, I^{"}_l, \cdots, I'_{q-1}, I'_{q+1}, \cdots, I'_s$], where all insertion operations are disjointed. So $\widetilde{OB}$ = **COMET**([$D_1, \cdots, D_r, I_1, \cdots, I_s, O_{n+1}$]) = [$D_1, \cdots, D'_k, \cdots, D'_r, I'_1, \cdots, I'_s$] or [$D_1, \cdots, D'_k, \cdots, D'_r, I'_1, \cdots, I^{"}_l, \cdots, I'_{q-1}, I'_{q+1}, \cdots, I'_s$], where all operations are disjointed. That is $\widetilde{OB} = \Omega_{OB}$ and the theorem holds.

3. Given $\forall D_i$ (1 $\leq i \leq r$), where $D_i \odot O_{n+1}$ but $\exists I_k$ (1 $\leq k \leq s$), where $I_k \oplus O_{n+1}$.
   (a) If O_{n+1} can be totally merged into I_k, $\widetilde{OB}$ = **COMET**([$D_1, \cdots, D_r, I_1, \cdots, I_s, O_{n+1}$]) = [$D_1, \cdots, D_r$, **COMET**([$I_1, \cdots, I_s, O_{n+1}$])] = [$D_1, \cdots, D_r, I_1, \cdots, I'_k, \cdots, I'_s$], where all operations are disjointed. That is $\widetilde{OB} = \Omega_{OB}$ and the theorem holds.
   (b) If I_k can be totally merged into O_{n+1}, $\widetilde{OB}$ = **COMET**([$D_1, \cdots, D_r, I_1, \cdots, I_s, O_{n+1}$]) = **COMET**([$D_1, \cdots, D_r, O'_{n+1}, I'_1, \cdots, I'_{k-1}, I'_{k+1}, \cdots, I'_s$]) = [$D_1, \cdots, D_r, O'_{n+1}$, **COMET**([$I'_1, \cdots, I'_{k-1}, I'_{k+1}, \cdots, I'_s$]) where D_i (1 $\leq i \leq r$) and O'_{n+1} are disjointed deletion operations. If O_{n+1} creates a new adjacent relation between I_l and I_q (1 $\leq l, q \leq s$), **COMET**([$I'_1, \cdots, I'_{k-1}, I'_{k+1}, \cdots, I'_s$]) = [$I'_1, \cdots, I'_{k-1}, I'_{k+1}, \cdots, I^{"}_l, \cdots, I'_{q-1}, I'_{q-1}, \cdots, I'_s$], where all insertion operations are disjointed. So $\widetilde{OB}$ = [$D_1, \cdots, D_r, O'_{n+1}, I'_1, \cdots, I'_{k-1}, I'_{k+1}, \cdots, I'_s$] or [$D_1, \cdots, D_r, O'_{n+1}, I'_1, \cdots, I'_{k-1}, I'_{k+1}, \cdots, I^{"}_l, \cdots, I'_{q-1}, I'_{q-1}, \cdots, I'_s$], where all operations are disjointed. That is $\widetilde{OB} = \Omega_{OB}$ and the theorem holds.
   (c) If I_k and O_{n+1} can be partially merged, $\widetilde{OB}$ = **COMET**([$D_1, \cdots, D_r, I_1, \cdots, I_s, O_{n+1}$]) = **COMET**([$D_1, \cdots, D_r, I_1, \cdots, I'_k, O'_{n+1}, I'_{k+1}, \cdots, I'_s$]) = **COMET**([$D_1, \cdots, D_r, O^{"}_{n+1}, I'_1, \cdots, I^{"}_k, \cdots, I'_s$]) = [$D_1, \cdots, D_r, O^{"}_{n+1}$, **COMET**([$I'_1, \cdots, I^{"}_k, \cdots, I'_s$])], where D_i (1 $\leq i \leq r$) and $O^{"}_{n+1}$ are disjointed deletion operations. If O_{n+1} creates a new adjacent relation between I_l and I_q (1 $\leq l, q \leq s$), **COMET**([$I'_1, \cdots, I^{"}_k, \cdots, I'_s$]) = [$I'_1, \cdots, I^{"}_k, \cdots, I^{"}_l, \cdots, I'_{q-1}, I'_{q+1}, \cdots, I'_s$], where all insertion operations are disjointed. So $\widetilde{OB}$ = [$D_1, \cdots, D_r, O^{"}_{n+1}, I'_1, \cdots, I^{"}_k, \cdots, I'_s$] or [$D_1, \cdots, D_r, O^{"}_{n+1}, I'_1, \cdots, I^{"}_k, \cdots, I^{"}_l, \cdots,$

I'_{q-1}, I'_{q+1}, $\cdots$, I'_s], where all operations are disjointed. That is $\widetilde{OB} = \Omega_{OB}$ and the theorem holds.

4. If $\exists D_k$ (1 $\leq k \leq$ r) and I_p (1 $\leq p \leq$ s) where $D_k \oplus/\ominus O_{n+1}$ and $I_p \oplus O_{n+1}$, it can be deduced from (2) and (3) that operations in $\widetilde{OB}$ = **COMET**([D_1, $\cdots$, D_r, I_1, $\cdots$, I_s, O_{n+1}]) are also disjointed. That is $\widetilde{OB} = \Omega_{OB}$ and the theorem holds.

By the induction argument, the theorem holds, that is, for an OB containing any number of operations, after **COMET**(OB), it must be $\widetilde{OB} = \Omega_{OB}$.

Due to space limitation, other important properties of the compression algorithm are not given in the paper.

5. CONCLUSIONS AND FUTURE WORK

Our approach to increasing the engagement of students in lectures using Smartphones is innovative, both in terms of the purpose that the device is being used for, but also the technical contributions allowing for the different learning abilities and styles. A single editor, where notes from all group members are intermingled and there is no opportunity for an individual to look at just a single group member's notes, does not meet the primary requirement for flexible and effective collaborative learning in lectures.

Design for multiple editors with each being explicitly owned by a student but allowing for collaborative editing of the same note provided the driving force behind the technical innovations outlined in this paper. The innovative collaborative interface allows any group member to freely write into any editor at any time, the optimistic note synchronisation solution allows a note to be fully synchronised without requiring much network bandwidth or any constraint on users' activities, and the novel operation buffer compression algorithm allows the App to take full advantage of the available wireless networking resources and a Smartphone's battery life.

Future work includes usability study of the $GroupNote$ system, including evaluation of the user interface and collaboration features as well as study of human factors. Studies into collaborative learning using the system will be carried out to determine the learning outcomes of students participating in the system compared to either non-participation or participation using alternative technologies.

6. REFERENCES

Bligh, D.A. (2000) *What's The Use of Lectures?* San Francisco, CA, Jossey-Bass.

Falkner, K. and Munro, D.S. (2009) Easing the Transition: A Collaborative Learning Approach. *The 11th Australasian Computing Education Conference*, 65-74.

Hitchens, M. and Lister, R. (2009) A Focus Group Study of Student Attitudes to Lectures. *The 11th Australasian Computing Education Conference*, 93-100.

Kam, M. et al. (2005) Livenotes: A System for Cooperative and Augmented Note-taking in Lectures. *ACM Conference on Human factors in computing systems*, 531- 540.

Litchfield, A. et al. (2009) Using Students' Devices and a No-to-Low Cost Online Tool to Support Interactive Experiential mLearning. *9th IEEE International Conference on Advanced Learning Technologies*, 674-678.

Matheson, C. (2008) The Educational Value and Effectiveness of Lectures. *The Clinical Teacher* 5(4): 218-221.

Reilly, M. and Shen, H. (2011) GroupNotes: Encouraging Proactive Student Engagement in Lectures through Collaborative Note-taking on Smartphones. *The 9th International Conference on Computer Supported Collaborative Learning*.

Reilly, M. and Shen, H. (2011) Shared note-taking: A Smartphone-based Approach to Increased Student Engagement in Lectures. *The 11th International Workshop on Collaborative Editing Systems in conjunction with ACM Conference on Computer Supported Cooperative Work*.

Shen, H. and Sun, C., (2002) Flexible Notification for Collaborative Systems. *ACM Conference on Computer Supported Cooperative Work*, 77-86.

Simon, B. et al. (2008) Noteblogging: Taking Note Taking Public. *The 39th SIGCSE Technical Symposium on Computer Science Education*, 417-421.

Sun, C. and Ellis, C., (1998) Operational Transformation Rn real-time Group Editors: Issues, Algorithms, and Achievements. *ACM Conference on Computer Supported Cooperative Work*, 59-68.

Sun, C. et al. (1998) Achieving Convergence, Causality Preservation, and Intention Preservation in Real-time Cooperative Editing Systems. *ACM Transactions on Computer-Human Interaction*, 5(1): 63-108.