CoVim: Incorporating Real-time Collaboration Capabilities into Comprehensive Text Editors

Bryden Cho, Agustina Ng, and Chengzheng Sun
School of Computer Science and Engineering, Nanyang Technological University, Singapore
DCho003@e.ntu.edu.sg, {Agustina, CZSun}@ntu.edu.sg

***Abstract* – Collaboration among programmers with different expertise and skill sets is becoming highly essential as many software systems are getting larger and more complex. The *CoVim* project aims to incorporate advanced real-time collaboration capabilities into Vim, one of the most popular source code editors, using the *Transparent Adaptation* (*TA*) approach. This work has extended *TA* from GUI-based applications (e.g. *MS Word*, *MS PowerPoint*, and *Autodesk Maya*) to complex keyboard–oriented applications like Vim. This work also contributed a novel *state-centric operation derivation* approach for TA, which is generic and more efficient in supporting a range of real-time editing features, compared to prior TA's *interception-centric* operation derivation approach which is user-interaction dependent and application specific.**

Keywords – Computer-supported cooperative work, Operation Transformation, Transparent Adaptation, Collaborative Editing

I. INTRODUCTION

In this Information Age, computer systems are ubiquitous in our daily tasks and activities. Computer systems are becoming larger and more complex that their often tightly scheduled developments require collaboration among multiple programmers with different expertise and skills. Because of that, *Agile* software development method that emphasizes on collaboration among programmers is becoming increasingly popular in the field of software development.

Rather than recreating applications from the scratch to support collaboration, several approaches that support collaboration using existing single-user applications have begun to emerge in recent years [2][3][7][8][11][12]. Such approaches allow users to use their familiar single-user applications such as *MS Word*[1] for collaboration. One such approach is the *Transparent Adaptation* (*TA*) [12], which is built upon the *Operation Transformation* (*OT*) technology [6][12][14]. TA has been successfully applied in the development of *CoWord* [12] for collaborative word processing, *CoPowerPoint* [12] for collaborative slide design, and *CoMaya* [2] for collaborative 3D digital media design.

Despite all those successes, TA is still in its young stage of development, there are still vast amounts of research possibilities available. In this work, we applied TA, which had been previously applied to only GUI-based applications, to keyboard-oriented applications. We have chosen Vim[2] as our target as it is one of the oldest and most widely used text editors, particularly for software developments. Vim is a comprehensive text editor, supporting not only simple text insert and delete, but also complex text manipulation using the built-in or user-defined commands, that are executed through different editing modes, including *normal*, *insert*, *visual*, *select*, *command-line*, and *ex-mode*. Moreover, Vim is also radically different than those GUI-based applications that have been targeted by TA in prior works: Vim is operated largely through keyboard but it also has some sophisticated GUI features such as automatic syntax highlighting and multiple windows view.

At the first glance, Vim may seem to be a much simpler application to be transparently adapted into a collaborative application compared to those GUI-based applications such as *MS Word*[1], *MS PowerPoint*[3], and *Maya*[4], considering Vim only has simple string texts without fancy objects and attributes, such as font styles, complex 2D/3D shapes, animation, etc. But it turns out that applying TA to Vim is non-trivial due to its comprehensive and complex editing features, the support of multiple windows, etc. By learning from past experiences from TA works and applying the techniques that are evolved and inspired by OT works, two viable working prototypes, CoVim1.0 and CoVim2.0, have been developed. And from CoVim2.0, a novel TA-based technique has been invented, which is more generic than prior TA techniques, not only within CoVim to uniformly support all complex editing in different modes, but also generic enough to be potentially directly applied to other keyboard-oriented applications, e.g. Emacs[5], SubLime[6], etc.

The rest of the paper is organized as follows. First, we introduce background on TA and OT in Section II. Then, we discuss CoVim1.0 in Section III, which is the first version of CoVim that follows the footsteps of prior TA techniques closely, and highlight the key issues and challenges faced. Afterward, we discuss CoVim2.0 in Section IV, which uses a novel and more generic TA-based solution to overcome those issues and challenges, and significantly simplify and reduce the work needed in incorporating advanced real-time collaboration capabilities into Vim. The TA approaches in CoVim are compared with prior related work in Section V. Finally, we summarize major findings and contribution of this research and discuss directions for future work in Section VI.

II. BACKGROUND ON OT AND TA

A. Basic OT Technique

OT is an established conflict resolution and consistency maintenance technique that is the cornerstone of TA-based and many other real-time collaborative editing systems. The basic

[1] https://products.office.com/en-sg/word
[2] http://www.vim.org/
[3] https://products.office.com/en-sg/powerpoint
[4] http://www.autodesk.com/products/maya/overview
[5] https://www.gnu.org/software/emacs/
[6] https://www.sublimetext.com/

978-1-5090-6199-0/17/$31.00 ©2017 IEEE

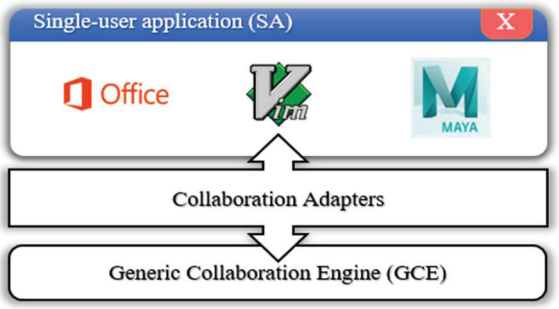

Fig.1. The Basic TA Architecture

idea of OT is to transform an operation defined on one document state into another operation on a new document state, so that the transformed operation can be correctly executed on the new document state and achieve document consistency in the face of concurrent operations [13][14]. OT-based applications are known to be able to perverse user generated operations effects in face of concurrency [13].

OT was originally designed to support collaborative editing of plain text documents, but has been extended and generalized to support collaborative editing of *MS Word* [12], 3D graphics [2], XML/HTML [6] documents, etc.

There are two underlying models in OT. The first one is the *data model*, which defines how data objects are addressed by operations. The second one is the *operation model*, which defines the set of operations that are directly transformable by OT. Different OT techniques have different data and operation models. As Vim is a plain-text editor, we use the basic OT technique [13]. In the basic OT, the entire document is treated as a string and each character is accessed linearly using its positional index in the string. All characters, including meta-characters like tabs and newline, are treated equally. There are two generic *Primitive Operations* (*PO*) in the basic OT [4][12]: (1) *Insert*(*p*, *l*, *s*) to insert string *s* of length *l* at position *p*; and (2) *Delete*(*p*, *l*, *s*) to delete string *s* of length *l* at position *p*. Complex editing operations can be represented using the combination of those two POs, e.g., a *Replace* operation can be represented by a *Delete* PO plus an *Insert* PO.

B. Basic TA approach

TA is an approach to incorporate advanced real-time collaboration capabilities, including high responsiveness, real-time notification, free and concurrent editing, and detailed workspace awareness, into existing applications without modifying their source codes [2][12]. The TA approach allows users to use their preferred and familiar, instead of separate and new, applications for collaboration. TA uses innovative techniques to build a *Collaboration Adaptor* (*CA*) that is transparent from the adapted single-user application and the basic OT system encapsulated in a *Generic Collaboration Engine* (*GCE*) as shown in Fig. 1. GCE is generic and shared among all TA-based systems. On the other hand, CA is application-specific and must be built for each single-user application that is to be transparently adapted.

There are two adaptation tasks in building a CA. The first one is the *data adaptation*, which is to examine how data objects are addressed by the single-user application from both UI and API, and to map this data model into the basic OT data model of

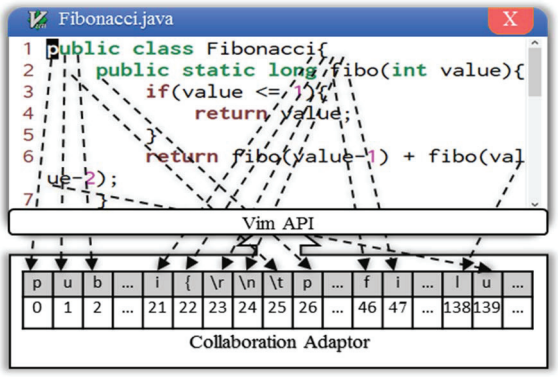

Fig.2. CoVim 's Data Adaptation

GCE. The second one is *operation adaptation*, which is to examine what and how operations are performed by the single-user application and to map each of those operations into POs so that they can be directly transformable by the OT system in GCE for consistency maintenance in the face of concurrent editing. These adaptations would involve using the Operating System's and/or single-user application's API to intercept and replay user operations. Intercepted user operations are represented as *Adapted Operations* (AO) for propagation to remote collaborating sites.

This TA technique has been successfully applied in various collaborative systems [2][12]. In the early stage of this work in building CoVim1.0, we followed this basic TA technique closely to do adaptation for each user interactions supported by Vim. Although CoVim can be built in this way, it was full of challenges mainly due to the need of deriving the effects of each user interactions in the operation adaptation, and the complex and comprehensive features Vim provides. This has motivated us to explore a radically different direction and invent a new TA technique in building CoVim2.0 that is generic and able to treat most if not all operations uniformly by using, instead of deriving, the final effects of user interactions produced by Vim in generating AOs. Both CoVim1.0 and CoVim2.0 will be discussed in details in the rest of this paper.

III. COVIM1.0

A. Data Adaptation

From the user's point of view, all characters in a Vim documents are presented sequentially in the user interface, and the same character can be presented in a different color, in a different width (for the tab character), and even in a different position horizontally and vertically under various settings.

However, all those different presentations are not the results of internal attributes of the character object, but rather caused by the syntax highlighting feature, the application settings (i.e. the tab width property in this case), and the adjustable window size of Vim. Therefore, they are not the concerns in the adaptation process, and in the CoVim's point of view, Vim's data object is just a stream of plain characters, without any attributes as shown in Fig. 2.

Vim provides rich and comprehensive APIs, not only for accessing and manipulating any data in the Vim document, but also for controlling the GUI appearance of Vim, such as the window size. From the API's point of view, every character in

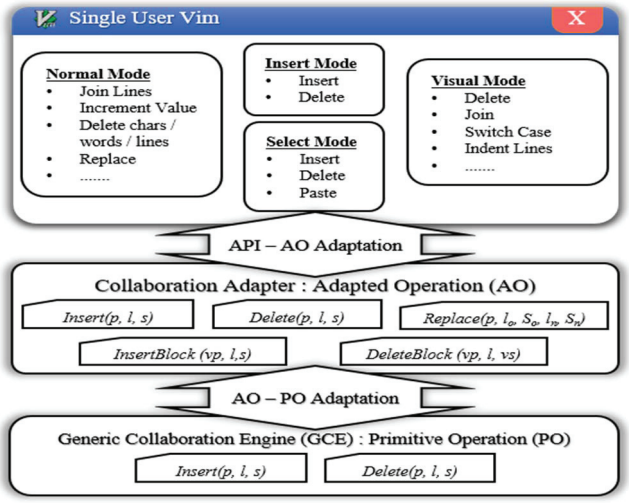

Fig.3. Three layers in CoVim Adaptation

the document has a position in the Vim's *line-column* data addressing model, and is addressed by using a (L, C) pair where L is the line number and C is the column number. For example in Fig. 2, the character '***f***' at the beginning of the word "***fibo***" is addressed by $(2, 21)$ as it is located at line *2* and column *21*. This line-column model allows Vim to run efficiently on different Operating Systems that may represent a newline differently, e.g. '***r*****n**' in Windows and '***n**' in Unix.

This *line-column* data model does not match the linear data model of the basic OT in GCE. To be able to take advantage of GCE for consistency maintenance, CoVim uses the Vim API *line2byte* and *byte2line* for mapping between (L, C) in the Vim's *line-column* data model and position P in the GCE's linear data model. To map from (L, C) to P, first get the linear position of the first character in line L using *line2byte*(L) and then add $(C - 1)$ into it (minus 1 because OT's position starts from zero but Vim's column number starts from 1). This can be expressed by $P = line2byte(L) + C - 1$. For the previous example, the linear position of $(2, 21)$ is $P = 26 + 21 - 1 = 46$ where 26 is the result of *line2byte*(*2*). To map from P to (L, C), first get the line number L where the character is using *byte2line*(P), then the column number C can be calculated by subtracting $P + 1$ with the number of characters preceding the line, i.e. *line2byte*(L). This can be expressed by $L = byte2line(P)$ and $C = P + 1 - line2byte(L)$.

B. Operation Adaptation

After the data adaptation, the next step in building CoVim is to adapt Vim operations into the operation model in GCE. As Vim is keyboard-centric, only key interactions that change the document state need to be intercepted and adapted into AOs for propagation and replayed at remote sites. But simply intercepting keyboard press events is insufficient because the same key events can have different meanings depending on the current editing mode. For example, the keyboard interaction of '***dd***' can mean deleting the current line if Vim is in the *normal* mode, and it can also mean inserting the characters '***dd***' into the document if Vim is in the *insert* mode. Therefore, the intercepted key events must be interpreted according to the mode it is operated on.

CoVim1.0 takes advantage of the *key-mapping* feature of Vim to overwrite the meaning of each possible key combination in each mode with CoVim's own procedures. User interactions that have similar behaviors regardless the Vim mode are adapted uniformly by the same CoVim procedure. When a user interaction is intercepted, CoVim will first derive the scope and effect of the interaction based on our functional knowledge of the application. CoVim then invokes *similar* Vim commands to achieve the original effect of the user interaction because the original function of the interaction has been changed by our key remapping. Afterward, CoVim will query Vim APIs for additional information about the new document state to generate the AO representing this interaction, and then propagate the AO to remote sites. Even though Vim has a large number of possible user operation, CoVim1.0 found that all supported interactions can be represented with the following five AOs as shown in Fig. 3:

1. *Insert*(p, l, s) to insert string s of length l at position p;
2. *Delete*(p, l, s) to delete string s of length l at position p;
3. *Replace*(p, l_o, s_o, l_n, s_n) to replace old string s_o of length l_o at position p with new string s_n of length l_n;
4. *InsertBlock*(vp, l, s) to insert string s of length l at multiple positions represented by the vector vp; and
5. *DeleteBlock*(vp, l, vs) to delete multiple strings of the same length l, represented by the vector vs at multiple positions represented by the vector vp.

For example, the user interactions to decrement the value of the terminating condition at line 3 in Fig. 2 from 1 to -1 by pressing the '***2***' key followed by '***CTRL*** **+** ***x***' in *normal* mode, are essentially replacing the string "***1***" of length 1 at position 78 with the string "**-1**" of length 2 from the CoVim's point of view, thus are represented by a *Replace* (*78*, *1*, *"1"*, *2*, *"-1"*) AO. For another example, the user interactions to extend the single tab character *"\t"* at the beginning of line 2 to 7 in Fig. 2 with an extra tab by first selecting line 2 to 7 in *visual* mode and then pressing the '>' key twice, are essentially inserting *"\t"* of length 1 at multiple positions from the CoVim's point of view, thus are represented by only an *InsertBlock* (*[26*, *64*, *83*, *101*, *106*, *147]*, *1*, *"\t"*) AO.

When remote sites receive the propagated AOs, CoVim first translates the remote AO into POs for OT consistency maintenance by GCE in the face of concurrency. The translation from an AO to POs is straightforward: the *Insert* AO is mapped into an *Insert* PO, the *Delete* AO is mapped into a *Delete* PO, the *Replace* AO is mapped into a *Delete* PO plus an *Insert* PO, the *InsertBlock* AO is mapped into multiple *Insert* POs, and the *DeleteBlock* AO is mapped into multiple *Delete* POs. The OT-transformed POs will then be directly interpreted by the means of Vim APIs for replay. As the Vim APIs to insert/delete characters operate based on the current caret position, caret maintenance is an important aspect: the caret must be temporarily moved to the remote operation position prior to execution and reposition based on the operation effect after the execution.

With the above data and operation adaptation, we have successfully built a working CoVim1.0 prototype, which has been demonstrated at multiple international venues [1].

C. Issues and Challenges

Even though CoVim1.0 is a working prototype supporting about 82% of Vim7's commonly used editing commands, the process of building CoVim1.0 was very challenging, mainly due to the choice of *interception-centric* operation derivation, i.e. intercept every keyboard interactions and derive the user intention and an AO for each interaction.

We realized that with this approach, CoVim would never able to fully support all editing features in Vim for the following reasons. Firstly, CoVim might theoretically be able to support all built-in commands in the complex and comprehensive Vim editor only if more time and human resources were allocated for operation adaptation. But, some of the coediting supports would no longer be working correctly in a newer version of Vim, should the semantics of existing commands changed or new commands added. Secondly, there are more than 14,000 external Vim plugins[7], which contains user's customized macros or recorded operations, publicly available, and many other privately developed and used. Those plugins may not only add new editing functionalities to Vim, which CoVim does not support, but also overwrite built-in Vim commands, which breaks the coediting support in CoVim.

The development and testing of those supported Vim commands was also challenging due to the requirement of deep and comprehensive functional knowledge of these commands. This functional knowledge is required not only for generating the correct AOs that would achieve the same effect at remote sites, but also for simulating the original effects of the command after being overwritten by CoVim for interception purpose. Often, this functional knowledge is incomplete due to the incomplete documentation and not accessing the Vim's source codes, which makes CoVim unable to achieve correct effects locally or remotely in some special cases.

Lastly, the interception and generation of AOs for each interaction is an overhead that affects the local responsiveness and real-time notification. This is particularly true even for the common and simple case of quick and continuous typing: the typing of *N* characters would invoke *N* interceptions, *N* AOs generation and propagation, at least *N* OT transformations, and *N* remote executions. CoVim1.0 has optimized the adaptation for this type of user interactions by buffering the typing intercepted and creating only one AO for them, hence CoVim1.0 is able to achieve a reasonably good local responsiveness and real-time notification. However, the additional CoVim buffering mechanism (of which details are beyond the scope of this paper) has its own extra complexities.

IV. COVIM2.0

A. Basic Idea of State-Centric Operation Derivation

The challenges due to the *interception-centric* TA-based approach in building CoVim1.0 have motivated us to seek a better alternative in applying TA to Vim. CoVim2.0 uses a *state-centric* approach, which is partly inspired by the success of using *differential* (*diff*) algorithms [5][10][15] in supporting collaboration such as *ICT2* [7] and *Flexible Diff* [11]. The main idea behind this new approach is to periodically create the *snapshot* of the current state of the Vim document and use a diff algorithm to derive the *Insert* and/or *Delete* operations that change the first snapshot into the second snapshot. These operations are called *Differential Operations* (*DO*), which replace the *AOs* in CoVim1.0 for propagation to remote sites. At remote sites, *DO*s, which are also a sequence of *PO*s, can be directly OT-transformed by GCE before replayed. This basic idea is illustrated in Fig. 4. This *state-centric* approach maintains the TA architecture as shown in Fig. 5, and isolates the change to only the *CA* without affecting the rest of the architecture. This allows us to focus solely on devising a new approach for operation interception and generation mechanisms in CoVim2.0 to solve the issues and challenges faced by CoVim1.0.

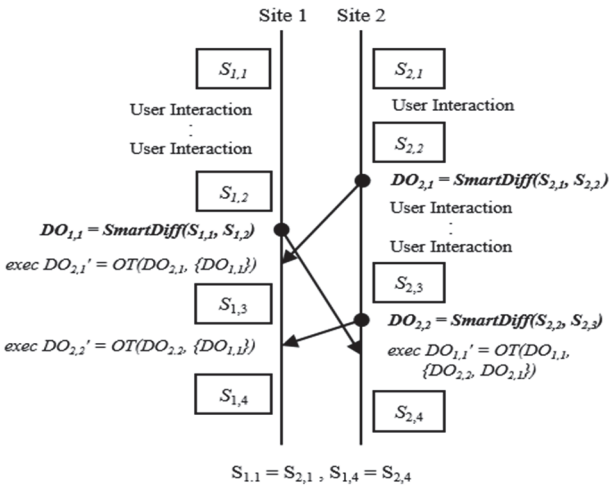

Fig.4. The basic idea of *state-centric* operation derivation

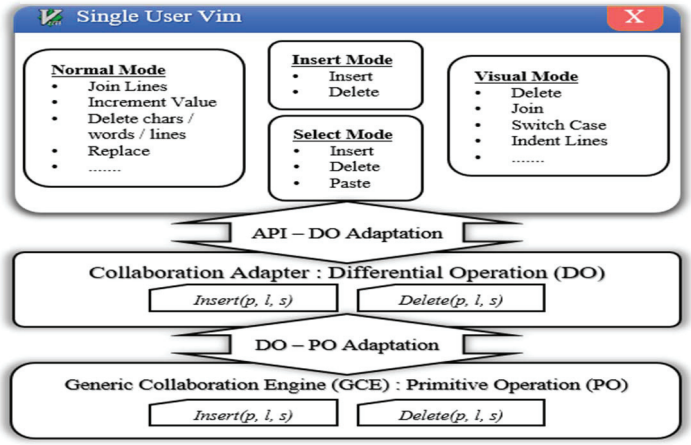

Fig. 5. CoVim 2.0 Architecture

This new approach focuses on the final effects of the user interactions and treats all user interaction uniformly, thus CoVim2.0 is able to support all built-in and user-defined commands without prior need of functional knowledge of these commands. And finally, this approach is more coarse-grained for text editing, such that with the correct frequency of snapshot generation and operation propagation, CoVim2.0 is able to achieve a better local responsiveness and real-time notification than CoVim1.0, without having to use an additional input buffering mechanism.

[7] http://vimawesome.com

B. Snapshots Generation

The diff algorithm for deriving *DOs* works by comparing two snapshots of the Vim document taken at different intervals. The snapshot, which is actually a string, also captures all meta-characters, such as *tab* and *newline*, in the Vim document so that the derived DOs can be correctly replayed at remote collaborating sites. The snapshot is generated by looping through the entire document line by line and concatenating them into a single linear string using the newline character of the targeted Operating System.

The frequency that snapshots are created, and thus the frequency that operations are generated, affects not only the real-time notification but also the local responsiveness because no user interaction can be generated during this process. To achieve local responsiveness that is as good as in the single-user Vim, while at the same time achieve real-time notification without consuming excessive network bandwidth, CoVim2.0 uses an *interactive polling* technique that dynamically changes the polling interval depending on the local user's activities, i.e. any user interaction will extend the next polling interval by one interval until an upper limit threshold is reached. The initial polling interval and the upper limit threshold were experimentally determined. A snapshot of the document must also be created after any successful remote execution, to be used as the initial snapshot at the next polling interval.

C. Differential Operation Generation

With the two recorded snapshots of the document, we can theoretically use any existing diff algorithm [5][10][15] to generate DOs. However, those algorithms are not fast and efficient enough for CoVim2.0 as they were not specifically designed to be used for real-time text editing systems. CoVim2.0 uses a novel differential technique, called *Smart Diff*, which is specifically designed and optimized towards real-time text editing. *Smart Diff* takes advantage of the characteristic of text editing: changes in the document tend to be adjacent to each other in most cases. Therefore, given two string of snapshots S_1 and S_2, we can find the common prefix S_p and common suffix S_s, which are the boundaries of the adjacent edits. For example, if S_1 = "***adoption***" and S_2 = "***adaptation***", we have S_p = "***ad***" and S_s = "***tion***" and the edits are in-between these two substrings, i.e. deleting the substring "***op***" in S_1 and inserting "***apta***" in S_2.

Let $len(S)$ denotes the length of a string S, we can derive the differential operations that change S_1 to S_2 as follows:

1. If the contents of the two snapshots are the same, i.e. $S_1 = S_2$, then there is no differential operation.
2. If the length of the first snapshot S_1 is not the same as the total length of the common prefix S_p and common suffix S_s, i.e. $len(S_1) \neq (len(S_p) + len(S_s))$, then the characters in-between the common prefix and suffix in S_1 must have been deleted, which can be represented by $Delete(p, l)$ where $p = len(S_p)$ and $l = len(S_1) - len(S_p) - len(S_s)$.
3. If the length of the second snapshot S_2 is not the same as the total length of common prefix S_p and common suffix S_s, i.e. $len(S_2) \neq (len(S_p) + len(S_s))$, then the characters in-between the common prefix and suffix in S_2 must be newly inserted, which can be represented by $Insert(p, s)$ where $p = len(S_p)$ and $s = substr(S_2, p, len(S_2) - len(S_s))$ that returns the substring from position p to $len(S_2) - len(S_s)$ in S_2.

The position of a DO is linear, which matches the OT data model and thus eliminates the need for additional mapping from the Vim's line-column model as required in CoVim1.0. Based on the above, there are four possible operation sets derived by Smart Diff: [ ], [*Insert*], [*Delete*], [*Delete, Insert*].

There is one special case in which the above basic *Smart Diff* would derive wrong DOs. This is when the common prefix S_p overlaps with the common suffix S_s because the substring of the characters inserted are identical to a sequence of characters directly before and/or after it. For example, suppose we have the S_1 = "***test***" with $len(S_1) = 4$ and S_2 = "***testset***" with $len(S_2) = 7$ after the user inserted "***set***" at position 4. *Smart Diff* will derive S_p = "***test***" with $len(S_p) = 4$ and S_s = "***t***" with $len(S_p) = 1$. Because $len(S_1) \neq (len(S_p) + len(S_s))$, i.e. $4 \neq (4 + 1)$ and $len(S_2) \neq (len(S_p) + len(S_s))$, i.e. $7 \neq (4 + 1)$, *Smart Diff* will output DO = [*Delete* (4, -1), Insert(4, 2, "**se**")], which is incorrect and different from the user input. To solve this issue, *Smart Diff* detects and corrects the common suffix: if the total length both of common prefix S_p and common suffix S_s is more than the minimum length of the two snapshots, i.e. $(len(S_p) + len(S_s)) > min(len(S_1), len(S_2))$, the length of common suffix is reduced by the overflow, i.e. $overflow = len(S_p) + len(S_s) - min(len(S_1), len(S_2))$ and $len(S_s) = len(S_s) - overflow$. In the previous example, the correction is $len(S_s) = 1 - 1 = 0$ because $overflow = 4 + 1 - min(4, 6) = 1$, which give corrected S_s = "" and correctly generates DO = [$Insert(4, 3,$ "**set**")].

It is worth pointing out that due to the sequential nature of text editing, the differential operations derived by Smart Diff match the actual user intention in most cases, except in cases where the user issues a complex command that makes non-adjacent changes to the document. Such commands include the *search and replace* command and the example for *InsertBlock* AO in Section III.B, which Smart Diff will derive [*Delete*, *Insert*] operations that cover not only those characters being replaced, but also all characters in-between the first and last replace. Although the differential operations for complex operations may not reflect the actual user intention, the differential operations still result in the correct document state at remote sites. Most importantly, the document consistency will still be maintained. In the future, we will optimize Smart Diff to better reflect the actual user intention in the complex editing scenarios.

V. Related Works

More commercial editors, such as *MS Word* and *Apple IWork*, are now supporting collaborative work. However, editing features that can be used collaboratively are still limited compared to their corresponding single-user versions[8]. For example, *MS Word* supports collaboration only on modern *MS Word* files type, but some invaluable features such as macro and track changes are currently not supported[9]. Some cloud storage systems, such as *Dropbox*, are also now supporting collaboration using users' preferred editors for files stored in their clouds. Although some collaboration awareness features are provided, those cloud storage only supports asynchronous collaboration: concurrent edits cannot be merged automatically,

but instead conflicted copies of the file will be created for users to manually merge[10].

TA, which is adopted in CoVim, is a *collaboration transparent* approach that is able to incorporate real-time collaboration capabilities into existing single-user applications. With TA, users are allowed to use their preferred applications for collaboration and concurrent works are preserved and merged automatically using OT [12]. Other collaboration transparent approaches are *Flexible JAMM* [3] and *ICT* (*Intelligent Collaboration Transparent*) [8], but they have limitations such as stricter requirements that are difficult for many applications to meet [12].

ICT2 [7] also uses a *state-centric* operation derivation approach, but uses a less efficient generic *Diff and Merge* algorithm to derive the operations to be propagated to remote sites. Concurrent operations are merged by sorting them according to their target positions relative to the same last synchronized state. To support synchronization at any time, ICT2 would need to use a sophisticated control algorithm [9] with a centralized server, which can be a bottleneck. In contrast, CoVim2.0 uses a more efficient *Smart Diff* algorithm to derive DOs and uses OT with control algorithm such as COT [14], which does not require a centralized server, for merging concurrent operations.

VI. CONCLUSION

In this paper, we reported our experiences and findings in transparently incorporating real-time collaboration capabilities into the complex and comprehensive Vim editor using the *Transparent Adaptation* (TA) approach. CoVim1.0 took the *interception-centric* operation derivation approach as done in prior TA works in converting *MS Word, MS PowerPoint, and Autodesk Maya*, into collaborative applications. This approach requires interception of each user interaction with the application to derive the intention of the user and to generate edit operations, which is application-specific and complex for applications that have a large number of built-in and user-defined commands like Vim. As such, we have devised a novel *state-centric* operation derivation TA approach, which focuses on the final effects of the user interaction and uses the state difference between two snapshots to derive the edit operations. This *state-centric* operation derivation TA approach is generic and simple, and has been successfully implemented in CoVim2.0 to uniformly support all Vim editing whether built-in or user-defined.

From this work, we have learned that the *state-centric* approach is a novel way to transparently incorporating collaboration capabilities into existing single-user applications. The novel text-editing *Smart Diff* algorithm to derive the edit operations from one snapshot of the document to another, is simple and generic and can be directly used in transparently adapting other text editors such as Emacs and SubLime. We hypothesize that these new techniques can be used to support *heterogeneous collaborative editing* across different editors. We are exploring this in our ongoing work to support collaborative pair-programming in any environment. Other important ongoing works are optimizing Smart Diff to derive better operations for complex operations such as *Search and Replace*, and formal verification of the correctness of Smart Diff in deriving any user interactions.

[8] Reader may refer to Apple's Article at https://support.apple.com/en-sg/HT206181

[9] Reader may refer to Microsoft's Article at https://support.office.com/en-us/article/Document-collaboration-and-co-authoring-ee1509b4-1f6e-401e-b04a-782d26f564a4

[10] https://www.dropbox.com/en/help/7683

ACKNOWLEDGEMENT

This research is partially supported by an Academic Research Grant (MOE2015-T2-1-087) from Ministry of Education Singapore. The authors wish to thank anonymous reviewers for their insightful and constructive feedback.

REFERENCES

[1] Agustina, S. Sari, N. Hilda, and C. Sun, "Issues and experiences in developing CoVim: a professional real-time collaborative editor based on Vim," *The Thirteenth International Workshop on Collaborative Editing Systems* in conjunction with *ACM 2013 Conf. Comput. Support. Coop. Work – CSCW2013.*

[2] Agustina, F. Liu, S. Xia, H. Shen, and C. Sun, "CoMaya: incorporating advanced collaboration capabilities into 3d digital media design tools," *Proc. ACM 2008 Conf. Comput. Support. Coop. Work – CSCW'08* , pp 5, 2008.

[3] J. Begole. M. Rosson, and C. Shaffer, "Flexible collaboration transparency: Supporting worker independence in replicated application sharing systems," *ACM Trans. Comput.-Human Interact.* 6, 2, pp. 95-132, 1999.

[4] C.A. Ellis. and S.J. Gibbs, "Concurrency control in groupware systems," *Proc. ACM 1989 Conf. Management of Data*, pp. 399-407, 1989.

[5] N. Fraser, "Differential Synchronization," *Proc. 9th ACM Symp. Doc. Eng*., pp. 13–20, 2009.

[6] C. Ignat and M. Norrie, "Tree-based model algorithm for maintaining consistency in real-time collaborative editing systems," *Fourth International Workshop on Collaborative Editing* in conjunction with *ACM Conf. Comput. Support. Coop. Work*, 2002.

[7] D. Li and J. Lu, "A lightweight approach to transparent sharing of familiar single-user editors," *Proc. ACM 2006 Conf. Comput. Coop. Work – CSCW'06*, pp. 139–148, 2006.

[8] D. Li and R. Li, "Transparent sharing and interoperation of heterogeneous single-user applications," *Proc. ACM 2002 Conf. Comput. Support. Coop. Work – CSCW'02*, pp. 246–255, 2002.

[9] R. Li, D. Li, and C. Sun, "A time interval based consistency control algorithm for interactive groupware applications," *IEEE Conf. on Parallel and Distributed Systems*, pp. 429-436, 2004.

[10] E.W. Myers, "An *O*(*ND*) difference algorithm and its variation," *Algorithmica I*, pp. 251-266, 1986.

[11] C. M. Neuwirth, R. Chandhok, D. S. Kaufer, P. Erion, J. Morris, and D. Miller, "Flexible Diff-ing in a collaborative writing system," *Proc. ACM 1992 Conf. Comput. Coop. Work - CSCW '92*, no. November, pp. 147–154, 1992.

[12] C. Sun, S. Xia, D. Sun, D. Chen, H. Shen, and W. Cai, "Transparent adaptation of single-user applications for multi-user real-time collaboration," *ACM Trans. Comput. Interact.*, vol. 13, no. 4, pp. 531–582, 2006.

[13] C. Sun, X. Jia, Y. Zhang, Y. Yang, and D. Chen, "Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems," *ACM Trans. Comput. Interact.*, vol. 5, no. 1, pp. 63–108, 1998.

[14] D. Sun and C. Sun, "Operation context and context-based operational transformation," *Proc. ACM 2006 Conf. Comput. Coop. Work – CSCW'06*, pp. 279–288, 2006.

[15] Y.Wang, D. J. DeWitt, and J.Y. Cai, "X-Diff: An effective change detection algorithm for XML documents," *Proc. Conf. on Data Engineering*, pp. 519–530, 2003.