INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

Vector-Field Consistency para Trabalho Cooperativo

Vector-Field Consistency for Cooperative Work

João Filipe Ferreira da Costa

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática e de Computadores

Outubro 2010

Acknowledgements

First, I would like to thank my advisor, Prof. Luís Veiga, for all the hours invested in the correction of my words. Also, I would like to thank him for the guidance, words of advice and for the encouragement to take risks. Without these, this work would certainly not be finished by now.

I would also like to thank all my friends for all the invitations to impossible to attend events and for sharing with me all the pictures and stories of the most amazing moments that I could not witness. To my closest friends, 1000 '*thank you*'s for putting up with me. I know it is not easy. Pessoa louca, estás a ler?

Finally, I want to specially thank my brother for his cooperation in the reviewing and typing of some parts of this document: You're the man! Agora sim, para acabar, Mãe, Pai, só para facilitar, até vos escrevo em português, muito muito obrigado.

Lisboa, October 18, 2010
João Costa

to Ma, Pa, Bro & Mo

Resumo

Embora o trabalho cooperativo seja utilizado diariamente um pouco por todo o lado, as ferramentas de trabalho cooperativo não estão a ser utilizadas como o seu potencial permitiria adivinhar. Ainda que as razões por detrás deste facto não sejam totalmente claras, a verdade é que os utilizadores vêem as actuais ferramentas de trabalho cooperativo, por vezes, como um entrave para as actividades do seu trabalho.

Para contornar este problema, exploramos dois conceitos: a noção de consciência da localização (locality-awareness) e a noção de modelo de consistência contínua. A primeira refere-se à habilidade de uma técnica em ter em consideração a localização dos utilizadores para tomar decisões. A segunda corresponde à noção de uma solução de meio-termo entre disponibilidade e consistência. Estas duas noções, quando combinadas, dão origem a uma técnica capaz de definir os limites de divergência para cada utilizador em função da sua localização nos objectos replicados. Por exemplo, se tomarmos em consideração o cenário de trabalho cooperativo, podemos decidir em que actualizações cada utilizador está mais interessado e quais ele não necessita de estar consciente naquele momento. Desta forma, teremos uma experiência melhorada sem comprometer a performance geral da aplicação.

Esta mistura entre as noções de consciência de localização e modelos de consistência contínua é exactamente o que faz o modelo de Vector-Field Consistency (VFC). No entanto, este modelo foi projectado para o ambiente de jogos distribuídos. Neste trabalho, propomos o VFC para Trabalho Cooperativo, nomeadamente trabalho cooperativo baseado em documentos. Esta adaptação consistiu na redefinição dos conceitos de VFC, como a localização de utilizadores (pivot), distância entre pivots e objectos replicados, e zonas de consistência. A definição da estrutura do documento foi um passo essencial na definição destes conceitos.

Neste trabalho, estudamos primeiro o estado da arte da manutenção de consistência em ferramentas de trabalho cooperativo. Algumas das técnicas estudadas foram usadas para criar um sistema baseado em VFC que é capaz de aplicar diferentes limites de divergência para cada utilizador em função da sua localização no objecto replicado. Este modelo foi depois especificado para um documento Latex, que tem uma estrutura de documento suficientemente rica. Mais tarde, o modelo de VFC para Trabalho Cooperativo foi aplicado a um editor de Latex que foi previamente adaptado para incorporar funcionalidades de cooperação num modelo totalmente consistente. Finalmente, este protótipo foi utilizado para obter resultados que permitam definir o sucesso deste trabalho e os méritos e falhas deste modelo.

Abstract

Although cooperative work is widely used in a daily basis, nonetheless cooperative work tools are not being used as their potential would suggest. Even though the reasons behind this fact are not completely clear, the truth is that users, sometimes, see current cooperative work tools as a burden to their working activities.

To work around this problem, we exploit two concepts: the notion of locality-awareness and the notion of continuous consistency model. The first refers to the ability of a technique to take in consideration the location of users in the making of decisions. The second corresponds to a notion of a mid-term solution between strong and weak consistency, which controls the divergence limits to find the most adequate balancing between availability and consistency. These two notions, when combined enable a powerful technique which bounds divergence limits on a per-user basis depending on the location in the replicated objects. For example, if we take the cooperative work scenario in consideration, we can decide which updates each user is most interested in and which ones he should be unaware of for the moment. This way, we will have an improved experience without compromising the overall application performance.

This mixture between the notions of location-awareness and continuous consistency models is exactly what the Vector-Field Consistency (VFC) model does. However this model was designed for the environment of distributed ad-hoc gaming. In this work we propose VFC for Cooperative Work, an adaptation of the original VFC model to the environment of cooperative work, namely document-based cooperative work. This adaptation consisted in the redefinition of the VFC concepts, such as users' location (pivot), distance between pivots and replicated objects, and consistency zones. The definition of the structure of the document is an essential step to the definition of these concepts.

In this work, we first study the state of the art of consistency maintenance in cooperative work tools. Some of the studied techniques were used to create a VFC powered system which is capable of enforcing different divergence limits for each user depending on their location in the replicated object. This model was then specialized for a Latex document, which has a sufficiently rich document structure. Later, the VFC for Cooperative Work model was applied to a Latex editor which was previously enhanced to support totally-consistent cooperation capabilities. Finally, this prototype was used to retrieve results to define the success of this work and the merits and flaws of this model.

Palavras Chave
Keywords

Palavras Chave

Vector-Field Consistency

Modelo de Consistência Contínua

Consciência da Localização

Trabalho Cooperativo

Baseado em Documentos

Representação TreeDoc

Keywords

Vector-Field Consistency

Continuous Consistency Model

Locality-Awareness

Cooperative Work

Document-Based

TreeDoc Representation

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In everyday life, it is more and more common to perform *cooperative*[1] *work* to carry out some task. For example, the cooperative production of documents (e.g. articles, presentations, financial reports) is a daily chore in enterprises [25]. Also, wikis, the massive cooperative editor by excellence, are between the most used platforms on the Internet.

If writing is always a long and complex process, *cooperative writing* is an even more complex and difficult process. Cooperatively writing a text can indeed shorten the duration of the writing task if teams work well and so members do not hinder each others work. However, if on the contrary teams do not function well by themselves, the additional effort needed to make them work may not pay off.

On contrary to what may be general belief, *cooperative work tools* are not as used as much their potential suggests [25]. Causing this, may be the general impression that these tools represent more of a burden than a relief. Although this is true in some cases, the major obstacle is still the reluctance of experienced users in changing their everyday tools and methods of collaboration.

A suggested approach for cooperative tools to achieve the desired success is the idea of enhancing the existing work tools to supply them with functionalities addressing cooperation. Even though this may seem a promising technique, it is also a dangerous one. First, because modern mature work tools have so many functionalities, adding extra ones can compromise the tools usability. Also, due to the great complexity of the existing functionalities, it is not always obvious how to provide them with cooperation faculties.

Although investigation has not yet come to a conclusion about the best method for designing a successful cooperative tool there are already some good examples. Some popular examples of success are the ones of *Google Docs* and *Google Spreadsheets*.[2] Forgetting about the fact that they benefit from the popularity of their developer company, the key to success may lay in their simplicity. With not so many functions but with very familiar ones, these tools manage to create a good developing environment. Another example is of course *Wikipedia*, the most popular wiki tool in the world, where users collectively produce text at a continuous rate of 10 words per second.[3] Again, the simplicity and easy access of this tool are without a doubt reasons for its triumph. On the other hand, we have the *Google Wave*[4]

[1]Although some authors suggest that there is in fact a difference in this field between the terms *cooperation* and *collaboration*, in this work there will be no distinction.
[2]http://docs.google.com/
[3]http://en.wikipedia.org/wiki/Wikipedia:Statistics
[4]http://wave.google.com/

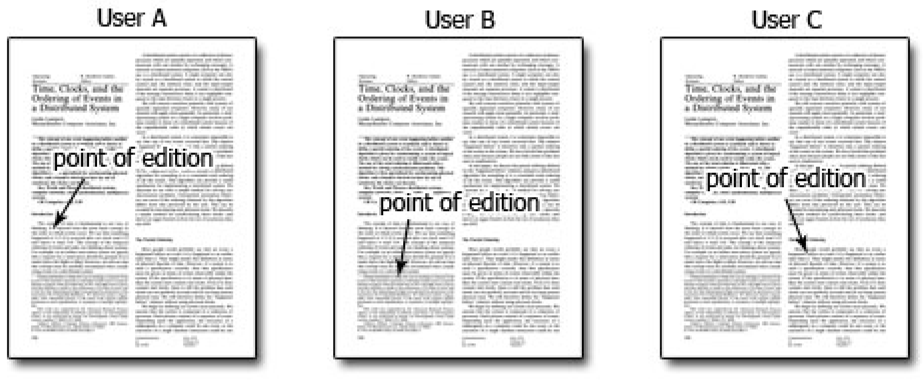


Figure 1.1: Typical example of cooperative edition of a document

example, which shows a completely different idea of what a cooperative tool should enable. Additionally to the cooperative work *per se*, Google Wave provides easy communication channels between members. The success of this project was not even close to the expectations which can be perhaps explained by the overcomplexity of the tool.

Although cooperative tools are little by little becoming more accepted and users are starting to see their potential, the *replication schemes* behind popular tools are still very conservative. Following the example of Wikipedia, it is implemented above a roughly centralized infrastructure and with a very restrictive conflict resolver mechanism. More, supporting such infrastructures can have huge costs when scaling these tools to a great number of users [27, 46, 22, 26].

Nowadays, popular tools do not try to do an intelligent management of updates. Instead, these tools use mostly all-or-nothing approaches. However, they could try to reason about the importance of each update and perform a selective scheduling based on this importance. This way, the user experience would be improved, since he would not be hindered by frequent but uninteresting updates. Also, since the selective scheduling would result in the delay of less important updates, savings in used network resources could be accomplished by merging of overlapping updates.

1.1 Contributions

This work proposes the adaptation of a consistency model more coherent with the cooperative tools paradigm. First, it proposes the enforcement of a *continuous consistency model* to better meet the requirements of these tools. Also, we discuss how, by combining the notion of *locality-awareness* with the continuous consistency model, we are able to adapt the consistency requirements to the current edition location of each user, within the document, and other points of registered interest. To implement these concepts, the *Vector-Field Consistency* [35] (VFC) algorithm was adapted and implemented to address the desired context.

Since VFC was designed to the gaming environment, it was necessary to port it to the world of document-based cooperative work, namely translate the concepts of location and distance between users locations, which are pretty straightforward in its original form. Following, *location* is defined as *the place in the document semantics where the user is editing* and *distance* as *the distance between the semantic regions being edited and other points of interest.*

An example of how the distance between users influences the consistency boundaries is the one given in Figure 1.1. In this picture, both user A and user B are editing the first section but on different paragraphs. User C on the other hand, is editing the second section. Thus, user A is probably working on a different subject than user C. As for users A and B, they must be careful because their writing about the same topic. Hence, consistency can be weakened between user C and the remaining ones, but strengthened between users A and B. This way, both A and B know with great precision what the other one is changing, but not what C is modifying.

1.2 Results

The main result taken from this work is the adaptation the Vector-Field Consistency algorithm to the cooperative work scenario, namely document-based cooperative work. However, other than the main theoretical artefact, in this work we produced the following:

- A survey and assessment of the related work in *consistency enforcement in distributed systems* and in *cooperative work*.
- A distributed version of the *Texmaker*[5] working with a total consistency model which, trough the use a commutative replicated data type, namely the *TreeDoc* representation, provides distributed a writing environment free of conflicts and loss of updates.
- An optimization of the distributed Texmaker which integrates the *VFC for Cooperative Work* model with increased performance results (w.r.t. network usage).
- An evaluation of the success of the implementation in terms of the following criteria:
  - *Qualitative*: conformity with the model, maintain user expected properties and functionality;
  - *Quantitative*: performance results, bandwidth usage;
  - *Comparative*: comparison with the total consistency version.

1.3 Document Roadmap

Next, in Chapter 2, we start by exploring the main issues related to consistency maintenance in distributed systems, and afterwards we overview the importance of cooperative work tools followed by a description of some examples. In addition to the overview of the state of the art of this area, the chapter provides a perspective of the benefits and flaws of various techniques which were or could have been used in this work.

Then, in Chapter 3, we detail the architecture that describes the solution. Later, in Chapter 4 some of the most relevant implementation aspects will be surveyed. Following, in Chapter 5, a series of quantitative, qualitative and comparative evaluation parameters are presented and their results analyzed. Finally, the Chapter 6 concludes this work and advances some future directions.

[5]An open-source Latex editor: http://www.xm1math.net/texmaker/

Chapter 2

Related Work

The development of cooperative distributed tools implies a need to share data across several replicas. In general, distributed systems, like cooperative tools, need to ensure the consistency of replicated data.

The following sections address the two principal themes studied to determine the state of the art of the Cooperative Work in Distributed Systems field. In section 2.1, the topic of *Consistency in Distributed Systems* [34] is presented. The differences and usefulness of a series of models and algorithms in this area are discussed. In section 2.2, some of the existing *Cooperative Work Tools* [32, 30] are introduced as well as their strengths and weaknesses. This set is mainly composed by tools and algorithms that focus on edition of either plain text files, structured documents or shared wikis.

2.1 Consistency in Distributed Systems

In a Distributed System, higher availability and performance is often accomplished using data replication. Data replication enables applications to work even when some replicas are unavailable. Moreover, several users can access data at the same time and without the need to contact some remote and possible busy location. The downside of data replication is the possibility of replica divergence/inconsistency. The balance between consistency and availability is a characteristic that separates systems in two opposite classes: *optimistic replication systems* [34] (section 2.1.2) and *pessimistic replication systems* (section 2.1.1). However, sometimes neither the pessimistic nor the unbounded optimistic approaches are acceptable to applications. Thus, it may be beneficial to explore the semantic space between the two alternatives. Hence, in section 2.1.5, the existing *continuous consistency model*, their virtues and flaws are presented. Finally, in section 2.1.6 the idea of mixing the state-transfer and operation-based approaches to provide efficient operation representations is overviewed.

2.1.1 Pessimistic Approach

Pessimistic replication systems provide single copy consistency by allowing only one user at a time to perform alterations to a replica. Therefore, the remaining users block until the first user finishes. This approach prevents the existence of conflicts and offer guaranties of strong consistency between replicas since the systems do not speculate whether it is safe or not to update data. Although the use of pessimist replication algorithms [5, 9] can work well in local-area networks, when ported to a wide-area network such as the Internet, these algorithms cannot provide good performance and availability.

Although according to a cooperative tools survey [25], content locking policies are among the most desired features for such tools, since pessimist policies are unable to provide the freedom of edition desired for this work, we will not further address the subject.

2.1.2 Design Choices in the Optimistic Approach

In opposition to Pessimistic techniques, Optimistic Replication assumes that conflicts will be extremely rare and can be fixed later whenever they appear. For this reason optimistic algorithms do not require *a priori synchronization* with the other replicas to perform an update. In result, these systems offer greater *availability*, *flexibility*, *scale better* and enable *asynchronous collaboration* even in wide-area environments. As for consistency, in optimistic algorithms, replicas may only converge eventually and so, controlling the differences between them in each moment is of the most importance. In result, these algorithms can only be deployed in systems that can support partially inconsistent data, even if only temporarily.

In optimistic replication [34], a series of design choices that define these optimistic systems can be considered. Those choices are defining characteristics that should reflect the consistency requirements of optimistic replication systems. They are: *number of writers*; *definition of operations*; *scheduling*; *handling of conflicts*; *propagation strategies and topologies*; *consistency guarantees.* The analysis of these criteria provides a great overview of the most important issues of optimistic replication.

Number of Writers: single-master vs. multi-master.

Single-master or *caching systems* are those in which only one replica can submit an update. Although easy to implement, these systems have very little availability. In comparison, *Multi- master* systems [49, 46, 28] are much more complex but offer greater availability. They let various replicas update content simultaneously and exchange the modifications in background. Multi-master systems, in contrast with the single-master, need to deal with the issues of scheduling and conflict handling.

Definition of Operations: state-transfer vs. operation-transfer.

Considering how modifications are propagated and exchanged between replicas, *State-Transfer* [36, 35] systems are a special class of systems that considers every operation to an object as a modification of the entire object. Although this may seem ineffective (and in fact, most times it is), there is some interesting potential behind this technique: for instance, replicas can easily converge by receiving the most recent contents without the need to apply every missing update. This use of the state-transfer approach can be very useful for example when a site stays offline for too long.

In opposition to State-Transfer systems, *Operation-Transfer* [11, 7, 37, 48] systems offer a more flexible conflict resolution and reduced bandwidth requirements at a cost of higher algorithmic complexity. To achieve that, these systems transfer one or more operations that correspond to the user changes instead of transferring the entire replicated object. This technique is particular useful when dealing with large and high level objects.

Nevertheless, when none of the above approaches meet applications requirements, hybrid solutions [23, 2, 44] may be useful. In section 2.1.6, the benefits and costs of such solutions will be analysed in more detail.

Scheduling: syntactic vs. semantic.

Eventual consistency is one of the most defining characteristics of optimistic replication. This concept means that all replicas will eventually come to the same value when users stop the commitment of new updates. To reach eventual consistency, replicas have to produce equivalent schedules (which produce equivalent final state). There are a number of systems that employ different types of scheduling. On one hand we have the *syntactic scheduling* [36] approach, which tries to define a total order of operations based on their submission timing and location. Coda [36] for example implements this type of scheduling using scalar timestamps.

On the other hand we have the *semantic scheduling* [39, 28, 27] approach, which exploits operation semantics to either transform (section 2.1.3) or commute operations (section 2.1.4). This approach has the advantage of reducing rollbacks and augmenting scheduling freedom in order to find the best (in the application point of view) possible schedule well-suited with the existing constraints. Since semantic scheduling requires systems to have access to some sort of semantic knowledge about objects, it can only be used in operation-transfer systems.

Conflict Handling: syntactic vs. semantic detection & manual vs. automatic removal.

Another important characteristic that separates various optimistic replication systems is the way they *handle conflicts*. Conflicts happen whenever operations do not respect their *preconditions*. Managing conflicts has two phases: *detection* of the conflict and *resolving* it. Numerous systems choose not to do any type of conflict handling, although at the cost of an increase in number of lost-updates and of truly difficult data management. As in scheduling schemes, also conflict management techniques cover the spectrum between syntactic and semantic approaches. *Syntactic conflict detection* finds conflicts through the use of the *happens-before* [20, 31] relationship. This means that every group of concurrent operations will be flagged as in conflict. *Semantic conflict detection* policies on the other hand use semantic information about the replicated objects to determine if there is in fact a conflict. This method, although not as efficient as the syntactic conflict detection, can benefit a lot from the lack of false conflicts.

Once a conflict has been detected, it is necessary to resolve it. In the resolution of conflicts, the objective is to rewrite or abort operations in order to remove conflicts and it can be either *manual* or *automatic*. Manually resolving conflicts is not much of a challenge since it simply presents two versions of the replicated object and let the user decide the final version. As for automatic resolution of conflicts, there are several methods to handle this: an interesting one is the Bayou [42] example, which attaches to each operation a precondition and a merge procedure. Then, before applying an operation, if the precondition is being violated, the merge procedure is executed and the necessary fix-ups preformed. Although this can be an appealing approach, studies showed that writing these merge procedures is truly difficult in most situations.

Repeatedly, a problem with conflict management is to determine when an operation is stable, that is, when its outcome result is no longer in a tentative state. This knowledge is useful for applications because, once an operation gets stable (committed) it can be removed from logs. In this context, *commitment* *protocols* play an important role. Some of these protocols apply what is generally called *agreement in background*; this solution however is based on vector clocks and equivalent structures and so do not scale well. Another much more complex solution is to apply a *consensus* algorithm to agree on the order in which operations are committed [8].

Propagation Strategies and Topologies: pushing vs. pulling & fixed vs. ad-hoc.

The propagation of updates can be analysed in terms of the *communication topology* and *degree of synchrony*. When dealing with fixed topologies, solutions can be extremely efficient, however, if the network is not structured or it changes dynamically the best solutions rely on *epidemic propagation* algorithms. Regarding the degree of synchrony, there are pull-based systems, i.e. systems in which each replica requests some set of the remaining replicas for their most recent updates, and push-based systems, i.e. systems in which each replica pro-actively sends their own updates to other replicas. Although this depends on each case, in most scenarios the quicker the propagation, the lower the conflict rate is.

Conclusions

A few conclusions can be extracted from this section: First, *Single-Master* systems are good for read-intensive or single-writer applications. Also, *Multi-Master State-Transfer* systems are fairly simple and have low memory requirements and so they are adequate for most replicated systems. Additionally, their bandwidth needs do not increase with the rate of updates since they can be easily merged into one. These systems have the disadvantage of not dealing well with conflict resolution because of their all or nothing update approach. The problems with semantically rich conflict resolution can be solved using *Multi-Master Operation-Transfer* systems, although there is a price to pay in terms of algorithmic complexity and log space overhead.

Following. Next, a review of the principal characteristics of the two major *semantic scheduling* schemes is made in sections 2.1.3 and 2.1.4. The first, the most mature of these techniques, is *Operational Transformation*. Then, *Operation Commutativity*, a younger but very promising technique.

Finally, two types of mixed approaches are presented: in section 2.1.5, an approach that offers the possibility of bounding consistency in a continuous spectrum between *pessimistic* and *optimistic* consistency; in section 2.1.6, a combination of the *transfer-based* and the *operation-based* approach, in order to achieve more efficient representations.

2.1.3 Operational Transformation

Operational transformation is a *semantic scheduling* technique originally designed for consistency maintenance in group editors. These systems have very specific constraints, namely short response times and support for free and concurrent editing (similar to *Google Docs & Spreadsheets*), which can be fulfilled by the use of this technique. Being such a complex technique, we will only address its defining characteristics and most important strengths and flaws. Hence, to fully understand how it works, a careful reading of the materials in the area is necessary.

The OT approach, pioneered by the *Grove* [11] system in the late *'80s*, consisted in the principle that an update should by immediately executed in the local replica after its creation and only then propagated to the remaining replicas. Then the remote replicas transform the operation right before its execution without needing to reorder the previous operations.

Grove made the contribution of identifying two inconsistency problems that would happen if operations were executed in remote locations without a previous transformation step and as soon as they arrive: *divergence* and *causality-violation*. First, considering that operations are not commutative between then, if executed in different orders, the final editing result will be different. This is the divergence problem. To understand the second problem, out of causal order execution, one can imagine the situation were replicas transmit their operations without synchronization. In this case, since operations are executed immediately after arrival, the execution order might not respect the natural causal order.

Grove then defined two correctness criteria to solve the previously identified consistency problems: (1) *convergence property*: all generated operations have been executed at all sites; (2) *precedence property*: if one operation O_a causally precedes another operation O_b, then at each site the execution of O_a happens before the execution of O_b. Grove system has two main components: the state-vector time stamping scheme for guaranteeing (2); and what became known as the *distributed OPerational Transformation* (dOPT) algorithm for ensuring (1). With the dOPT algorithm, every operation that obeys to the first criteria is transformed against an already executed operation which is independent with the first. This transformation should be done in such way that every site that executes the same set of operations produces the same end state.

Later, *REDUCE* [39] identified a third inconsistency problem, called *intention violation*, which consists on the execution of an operation that, if executed after another concurrent operation, may not change the content as expected by the user. This problem may seem similar to the divergence problem but, as proven by its solution, it is not. In fact, the divergence problem can be solved just by employing a serialization protocol, but the intention violation problem cannot. This happens because, even though replicas are consistent, the final state might not reflect the expectations of users. Considering the intention violation problem, REDUCE defined the subsequent consistency model, which is commonly called *CCI consistency model*:

A cooperative editing system is consistent if it always maintains the following properties:

- *Convergence*: when the same set of operations has been executed at all sites, all copies of the shared document are identical.
- *Causality-preservation*: for any pair of operations O_a and O_b, if $O_a \rightarrow O_b$, then O_a is executed before O_b at all sites.
- *Intention-preservation*: for any operation O, the effects of executing O at all sites are the same as the intention of O, and the effect of executing O does not change the effects of independent operations.

In opposition to Grove, in the REDUCE approach an undo/do/redo scheme is used for achieving convergence. As for intention-preservation, a new operation transform algorithm was applied, the *Generic Operational Transformation* (GOT) control algorithm. This consistency model and its implementation successfully solved the scenario in which Grove did not work.

An optimization to the GOT algorithm by the authors of Grove and REDUCE, called *GOT Optimized* (GOTO), can be found in [11]. This algorithm allows a similar approach to the one used in REDUCE but without the need for the undo/do/redo scheme. Also, by performing transformations on both the creation and execution contexts, this algorithm is able to reduce the number of transformations.

Since the appearance of the Grove and REDUCE systems, many other algorithms, optimizations and even scenarios of application were presented for Operational Transformation. Numerous systems, such as Jupiter [24], TreeOPT [16], CoWord [40, 49], MOT2 [7], UNO [47] and FEW [4] successfully applied this technique especially in the field consistency maintenance. An interesting extension to the original Operational Transformation scheme, which was designed by CoWord, is the addition of support for the update primitive (originally, all user alterations were translated to sets of insert and delete primitives). In MOT2, the OT approach was successfully ported to the P2P environment. In fact, with MOT2 any site can reconcile its copy at any time with any other site that owns a copy of the object while achieving overall convergence of copies. Finally, UNO not just ports OT to the P2P environment, it also proposes a new extension to support the undoing of any applied operation.

Although we found many optimizations and adaptations of the original works, there is an apparent limitation to the technique. The transformation algorithms described in these works all apply to similar scenarios were the replicated object has only textual contents. Thus, when we try to port these techniques to new environments with more complex replicated data types, we have no clear way of transforming the operations.

2.1.4 Operation Commutativity

In the cooperative editing environment, it is natural that replicas diverge if they do not execute operations in the same order. To address the problem of replica converge there are several techniques. One of them is the previously presented *Operational Transformation* (section 2.1.3) but, as said in [28], OT is too "*complex and error-prone*". An alternative solution, and the one in focus in this section, is *Operation Commutativity*, the condition that every pair of operations is in commutative relation.

Operation Commutativity aims to the automatic convergence of replicas, i.e. convergence without the need of any complex concurrency control (e.g. lock or serialization). To achieve automatic convergence, it is sufficient the use of a *Commutative Replicated Data Type* (CRDT), as it was baptised [37].

The CRDT approach considers that a document is formed as a sequence of atoms each univocally described by an identifier that does never change during the life span of a document. An atom can be any non editable element like a character or a graphics file. The space of identifiers must be dense and their total order must reflect the order of appearance of atoms in the document. These requirements suggest that rational or real numbers could be used as identifiers, however, they would require infinite precision which is not viable.

The *TreeDoc* [37, 28] is an implementation of the structure of identifiers based on binary trees that represent the document elements/atoms. The total order of those elements can be translated from walking the tree in infix order, which means that the identifiers can be obtained from the tree paths. Since binary trees cannot by themselves support concurrent edits, *(major) nodes* in the identification tree can contain several *mini-nodes* to represent those edits. These structures must be identified inside the major-node by a *disambiguator* which may be either a *unique disambiguator* (UDIS: $<site_{id}$, $site_{counter}>$) or a simple *site disambiguator* (SDIS: $<site_{id}>$). Even though the first may seem inappropriate since it has a greater overhead, the latter imposes the use of tombstones, which can only be safely deleted when all sites are alive [34]. To fully understand some of the former concepts a more careful reading of [37, 28] is advised.

The proposed TreeDoc algorithm for generating new identifiers can be improved by combining it with algorithms to balance the identifier tree.

Another replication mechanism that was developed earlier is presented in [33]. Although the solution is similar with the CRDT approach, it relies on tombstones and vector clocks and so it has problems to scale to massive collaboration environments. Additionally, this solution has the problematic issue of being destructive and losing work. Thus, it cannot be applied to cooperative environments.

One more system that implements operation commutativity is the *Wooky* [46] system which was based on the *Woot* [27] framework (Section 2.2.2). Even though this system was developed independently from TreeDoc, they have many similarities. In fact, Wook also bases its design in the idea of creating non-destructive operations with unambiguous identifiers that live as long as the document does. One significant difference between these two approaches is the way Woot structures the documents and identifiers: a linear structure stores elements in their order of appearance in the document, each one indentified by $<site_{id}$, $site_{counter}>$). Even more, Woot does not allow the removal of the identifiers of deleted elements. This results in a constant growth of the supporting data-structure.

The same authors that created Woot [27] and Wooky [46] later developed the *Logoot* [48] model. The objective of this model is to ensure CCI consistency (see section 2.1.3) and scalability both in number of users and updates so it can be adapted to massive collaboration environments (e.g. over a P2P network). In consequence, one key design goal of this model is not to be tied to tombstones because of their scalability and performance problems. Like the previous examples of TreeDoc and Woot, this model is based on non-mutable and totally ordered identifiers [48]. Since the identifiers are totally ordered, they can be removed without affecting the order or the remaining identifiers.

2.1.5 Continuous Consistency Models

Designers of replicated systems conventionally choose between strong and optimistic consistency models. Although sometimes, neither the performance overheads imposed by strong consistency neither the lack of limits for inconsistency are acceptable to applications. In such cases, it is appropriate to explore the semantic space between these two alternatives. The fundamental idea behind these *continuous consistency models* is that this space is a continuum parametrized by the *distance* between replicas. This distance is zero for strong consistency and infinite for optimistic consistency. The distance measure can be used to provide a per- replica consistency based on the expected amount of conflicting updates.

Leverage of the consistency space continuum allows systems to correctly balance applications availability and consistency. This balance is affected by factors like application workload, read/write ratios, probability of simultaneous writes, network latency, bandwidth, error rates, etc. Some of the work developed in this field is based on bounding consistency along a single dimension, e.g.: "*maximum time without being made consistent*"; "*maximum number of uncommitted updates*" [18]. Even though studying single dimension consistency bounding may be interesting, this section will focus in more comprehensive models [50, 35, 3] with greater expressive power.

The TACT framework

In [50] is presented a middleware called TACT, which enables applications to quantify their consistency requirements. Once those requirements are defined, the TACT framework only lets one operation proceed locally if the consistency bounds are not currently being violated. Otherwise, TACT blocks the operation and synchronizes with the remaining replicas until the fulfilment of the pre-defined consistency requirements.

With TACT, applications need to specify their *conits*, i.e. the physical or logical unit of consistency. For example, in an airline reservation service, the conit could be an individual flight, or a block of seats or even a single seat. Defining the granularity of conits depends solely on the application necessities. The quantification of divergence boundaries is made on a per-replica basis through the use of three metrics, *Numerical Error*, *Order Error*, and *Staleness*. The first metric, *Numerical Error* bounds the difference between the local value of the conit and the value of the 'final image', i.e. the image in which all tentative updates of all replicas have been applied. The implementation of this metric can be tricky since, if application conits do not represent a numerical value, a numeric representation (a weight) must be specified. Also, this metric relies on the cooperation of all replicas and thus, cannot be dynamically changed without a consensus like protocol. On the other hand, *Order Error* can be controlled using only local data, since it bounds the maximum number of tentative writes at a replica. In other words, it bounds the number of local writes that may still be rolled backed. Finally, *Staleness* is the maximum value of time between the last seen local write and the current time. To bound this metric each replica holds a real-time vector, in which each entry corresponds to the real time passed since the last seen update from each replica.

This model enables the definition of per-replica consistency bounds, which allows systems to greatly adapt to their consistency requirements. For instance, one replica with limited network access may relax its consistency limits. Oppositely, in a replica with faster links it may be viable to impose a stronger consistency. Another interesting application of TACT is the routing of clients to different replicas (with different divergence bounds) according to their profile. Balancing consistency to meet each replica needs [50, 51] can have serious impacts in system performances. In fact, most times, applications can have significant performance improvements without compromising correction by slightly relaxing consistency.

The Vector-Field Consistency model

Like the previously presented TACT framework, *Vector Field Consistency* (*VFC*) [35] is a consistency model that enables replicas to define their consistency requirements in a continuous consistency spectrum. Other than simply *bounding divergence* on a per-replica basis, VFC allows more powerful consistency enforcement policies. For example, using VFC, the maximum allowed difference between two replicas can be dynamically changed during the execution of an application.

Indeed, the novelty of the VFC model is that it combines divergence bounding with the notion of *locality-awareness* to improve the availability and user experience while effectively reducing bandwidth usage. To understand how locality-awareness affects this model, the concept of *pivot* must be introduced. Pivots roughly correspond to each user's observation points within the data and their position (w.r.t. some set of coordinates) is expected to be volatile. Also, consistency between replicas should strengthen as the distance between their pivots shortens. To define these mutable divergence bounds, around pivots there are several concentric ring-shaped *consistency zones* with increasing distance (radius) and decreasing consistency requirements (increasing divergence bounds). Then, in each zone, like in the TACT framework, programmers use a 3-dimensional vector: *time*, *sequence*, *value*. These boundaries should be specified in a way that does not compromise the user's experience. In other words, all the required information is presented to users with enough quality and they cannot perceive much difference in their use of the applications.

The VFC model is extremely flexible with regard to the possibility of specifying different consistency boundaries. This flexibility allows VFC to be used in a wide variety of systems with very different consistency enforcement policies. Moreover, VFC is simple and intuitive in such way that developers can easily express and parametrize their consistency requirements in accordance with the application requirements. Also, VFC effectively reduces the network bandwidth requirements by selectively choosing which updates are more important to which replicas and delaying less important ones, possibly omitting some superseded by later updates.

In comparison with VFC, the TACT framework can enforce the same consistency scenarios but, since it does not support locality-awareness, it cannot be applied to scenarios where consistency depends on the notion of each user position within the data.

Although Vector-Field Consistency was initially design to fit the environment of distributed game development, the concepts of pivots, consistency zones and distance between replicas are sufficiently general to be easily applied to other distributed environments like cooperative work tools based on shared data.

Data aware Connectivity

The interest about the concept of *data-aware connectivity* system [3] is the use it makes of a continuous consistency model. The continuous consistency model is used to determine the *quality* of a replicated object. Then, unlike in the previously presented works, using the ascertained current value of quality, the system *regulates connectivity* to enforce the divergence/quality bounds. Only when all connections were explored without success it forbids access to the replica.

This way, the connectivity costs can decrease because only the exactly required connections will be used to achieve the imposed quality, which can be very useful in environments with great constraints like mobile environments.

These systems have two key components: the *quality monitor* to estimate the quality of each available replicated object; the *connectivity regulator* to enforce a certain level of replica quality. Quality is influenced, among other criteria by its *freshness*, *consistency* and possibility of *rapid commitment*, which are represented by a group of variables: (1) time since the last synchronization from each replica; (2) number of local tentative updates; (3) commit weight of currently inaccessible replicas; (4) number of known concurrent updates; and (5) recent update activity by other replicas to the object. The first (1) variable represents how fresh the replica is. The second (2) can be used to determine consistency. The remaining (3-5) determine the probability of a quick and successful update.

Another important issue about being up to the system to determine replica quality is the fact that these metrics, although intuitive to the attentive user, are not easy to evaluate by the human user. Thus, it is less error-prone to let the system deal with connectivity issues and provide the user an indicative value for the quality of replica so he can decide if he wants to perform a certain task.

2.1.6 Efficient Update Representations

The *definition of updates* has two well-known solutions: *state-transfer* and *operation-transfer*. While the first can be very easily and transparently adapted to existing commercial solutions, the second solution promises high concurrency and lower conflict rates. The idea of mixing the two solutions to get a transparent and yet highly available middleware for distributed systems can be very attractive.

The proposal of *Chunks* [23, 2] and *Semantic Chunks* [44] is to exploit content similarity with the intention of reducing the *bandwidth and memory requirements* and in the case of semantic chunks the amount of *update-conflicts* due to false-sharing.

Chunks were implemented in the *LBFS* [23] and later in the *Haddock-FS* [2] with the primary objective of seriously reducing the occupied memory and bandwidth. To accomplish this goal, an *application transparent* middleware was developed, which takes advantage of cross-file and cross-version similarities. In these systems, every file is partitioned in several parts called chunks, which, in order to be resilient to insertions and deletions within the file, are divided by a content-based approach. By applying the *SHA-1* [10] function to each chunk, they can be unambiguously identified. Using these unique identifiers replicas can then create a chunk repository to reduce memory. Also, by previously exchanging these identifiers, replicas can determine exactly which chunks must be transferred, thus reducing data sent over the network. Due to the great network and bandwidth reductions achieved, the use of chunks allows distributed file systems to be ported to scenarios of slow or wide-area networks.

Semantic-Chunks on the other hand, are semantically richer than Chunks. Since they can be independently considered for consistency issues, an increase in concurrency and a reduction (in number and cost) of update conflicts is expected. The Semantic-Chunks System consists in a middleware in between the OS/VM and the Office Applications that stores the semantic structure of the documents requested by those applications. These semantic structures (semantic-chunks) are aggregations of either lower-level semantic-chunks or chunks of data. This way, it is possible to dynamically obtain the documents as their sub-regions are required by the system or as new updates are submitted. Whether the data chunks are paragraphs, sentences or even characters, it is not defined. This decision can even be made dynamically so the system can adapt to various consistency requirements. The semantic-chunks middleware is completely transparent to the above applications.

2.2 Cooperative Work Tools

Although the design of cooperative software is an ancient and well-studied field, being a comprehensive and somewhat, fairly subjective area, naturally it is not one of much consensus. To address this issue, this section will start with an overview of the positive and negative aspects of *cooperative writing* [41, 25] as well the desired properties of good *cooperative writing tools* [14, 25]. Later, the concept of *CSCW* and their core issues will be introduced [13, 14, 1, 32, 30]. Finally, some of the most relevant *distributed document editors* and *replicated wiki* systems will be examined (section 2.2.1 and 2.2.2).

Cooperative Writing

The process of writing an article, a review, a book or even a simple draft may be long and complex. To ease the workload and consequently reduce the time consumed by writing tasks and achieve better results, writers tend to group together. Also, sometimes is natural that the work chores, like participation in reporting committees or in a research cooperative project, involve projects of joint writing.

If any cooperative face-to-face project is difficult, due to the problems of team management, a cooperative writing project is even more since it usually involves people in remote locations working together in an almost independent way. In these scenarios, the lack of communication and acknowledgement changes of other writers can easily lead to problematic events like writing the same thing more than once and accidentally deleted contents.

Nonetheless, the cooperative writing has very encouraging benefits, namely: *reducing task completion time*; *reducing errors*; *combining different viewpoints and skills*; and *obtaining an accurate document* [41, 25]. Also, writers tend to feel more involved with the outcome of the project and thus, contributing with *more time and effort* to the writing process.

Cooperative Writing Tools: What users want?

Although, as discussed, group editing has many great benefits, cooperative writing tools are nowadays still used only in a small part of cooperative writing projects. Most collaborative writers use simple personal word processors [25].

This lack of usage of group writing tools is in part due to the idea that they difficult the cooperative tasks instead of helping, which can be in fact true because systems tend to be designed without taking in consideration how groups really collaborate. Also, experienced users tend to be extremely reluctant about changing their writing software.

To mitigate the existing inertia in changing from non-cooperative to cooperative tools, a suggestion is, instead of developing a cooperative writing editor, to integrate cooperation features into the already existing and utilized software. On the other hand, some authors suggest that adding cooperation features to the already huge set of features offered by these editors, would create such a mess that they would become useless [25].

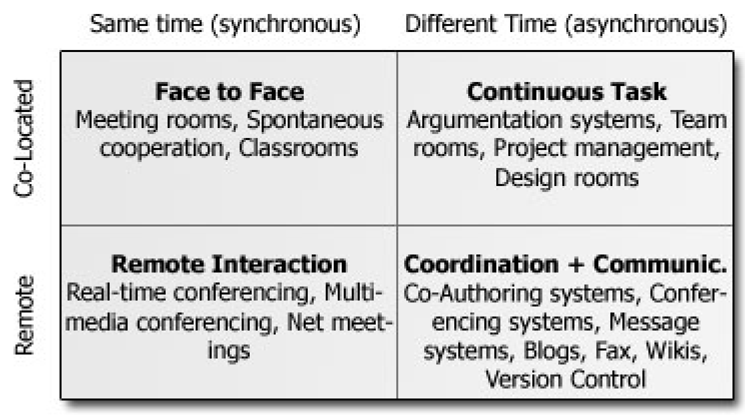
Enquired users [25] said that the cooperative features they would like to see in their writing tools were: *easy perception of recent modifications*; *easy addition of notes* (distinct from normal text); *incorporated communication channels*; and *text locking functions.* Curiously, on the opposing side, some users completely refused the idea of using a collaborative writing tool: "*The system is not nearly as important as the people with whom one writes. (...) I think a phone, a fax, or email are perfectly suitable*".

As we can see, investigators have not yet come to a conclusion about what should be a cooperative writing tool. There are already suggestions about interesting features but there are still no so many good examples of successful tools.

Computer Supported Cooperative Work (CSCW)

The idea of CSCW [13] was initially introduced by *Irene Greif* and *Paul Cashman* in 1984 to refer to the set of concerns about *supporting multiple individuals working together with computer systems.*

Nowadays, the term of CSCW is somewhat a confusing term. In [1] a simple analysis to the words of the acronym can explain the wide-coverage of this field. The expression *cooperative work* refers to



	Same time (synchronous)	Different Time (asynchronous)
Co-Located	**Face to Face** Meeting rooms, Spontaneous cooperation, Classrooms	**Continuous Task** Argumentation systems, Team rooms, Project management, Design rooms
Remote	**Remote Interaction** Real-time conferencing, Multi-media conferencing, Net meet-ings	**Coordination + Communic.** Co-Authoring systems, Confer-encing systems, Message systems, Blogs, Fax, Wikis, Version Control

Figure 2.1: Time/Space Matrix

multiple persons working together to produce a product or service. On the other hand, the second term, *computer supported*, does not limit so much the forms of interaction and organization. In result, CSCW is now a huge field that generically refers to the understanding of the way people work in groups and the associated network technologies, hardware, software, services and techniques.

As CSCW is a broad field, it may be beneficial to focus in just some part. Well, a term similar to the CSCW (some authors even considered them to be the same) is *Groupware*. Groupware is more focused on the enabling technologies which allow work in group, rather than the psychological, social, and organizational impact [30]. From now on the term CSCW will be referring to the same as Groupware, unless when addressing some specifics of one of them.

Due to the great variety of CSCW systems, to help distinguish the ones that matter from those that do not. An interesting division can be made by considering two key characteristics: *space* and *time* (Fig. 2.1). The first is about the *form of interaction*, i.e. if systems work synchronously or asynchronously. The second is about the *geographical nature* of the users, i.e. if working groups are co-located or remotely located. Using these distinction criteria, CSCW systems can be roughly divided in: *Message Systems*, *Computer Conferencing*, *Meeting Rooms* and *Co-Authoring and Argumentation Systems* (these are the designations presented in the '91 *Survey of CSCW System* [32]). However, nowadays this division does not create disjoint groups because, has systems evolved, so did their original characteristics.

Message Systems are an evolution from the electronic mail systems, from which they inherit the message exchange paradigm for cooperative work. Like in electronic email, message systems work *asynchronously* and *remotely*. The main difference between these systems and their ancestors is the evolution from a simple exchange of text messages to richer and structured message objects with specific attributes.

A *Computer Conferencing* system is a collection of *conferences/conversations*, each with a group of interested members and (desirably) a single topic. Also, usually each user can know what is new in each conversation he follows. Computer Conferencing started as simple *asynchronous* systems with shared textual information. They later *evolved to synchronous* systems to deal with real-time cooperative scenarios like crisis management. They also evolved to the so called multi-media conferencing systems, which due to network bandwidth improvements were able to integrate audio, text and video in the previous scheme.

Meeting Rooms are a special form of cooperative working paradigm where the participants work in a *synchronous* way and are *co-located* in a room equipped with information display technology and a workstation per member. Most of these systems were created for the proposed of decision making and for this reason they often incorporate statistical and analytic decision models. One interesting development was a multi-user interface with similar views for every participant. These solutions come although with a series of problems, like the distraction caused by the display of other participants activity and the high bandwidth requirements of such scenarios.

The objective behind *Co-Authoring and Argumentation Systems* [49, 44, 46, 26] is to support the development of co-authored documents. The idea is to have each author working on a portion of the whole document in an *asynchronous* manner *independently of their physical proximity.* This model of cooperation can be highly improved if dealing with structured documents since they permit easier isolation of user changes and conflict resolving. Also, as this field of investigation matured, it became obvious that the initial solutions where every user accessed a shared storage system did not work and decentralized solutions were established. In consequence the notion of private and master views/contexts appeared.

Although the proposed classes of systems were intended to divide systems, in most cases systems do not belong uniquely to one class. In fact, the most common situation is that these cooperative work systems combine a variety of characteristics and functionalities from each of the classes presented.

Following. In the next sections some interesting CSCW systems related to text edition will be introduced. In section 2.2.1, *simple document edition systems* will be analysed. Later, the section 2.2.2 will rather focus on *distributed wiki systems.* Finally, in section 2.2.3 we will discuss the importance of *intra and inter-document distance* in cooperative writing tools.

2.2.1 Distributed Document Editors

The Transparent Adaptation approach: CoWord

Single-user computer applications (text editors, graphics drawing tools, word processors, spreadsheets, presentation tools, web design tools, etc.) are widely present in our daily lives and work. Thus, since the concept of cooperative work is raising its visibility, it is natural to think about adding cooperation support to these applications.

Although the existing cooperation techniques were vastly studied by researchers, they were only applied to prototypes. Hence, it still remains to be determined the true applicability of such solutions in daily used mature applications. In [49] the authors propose how this leveraging between mature single-user application and state-of-the-art cooperative techniques can be carried out.

The continuous improve of current cooperative prototypes to achieve the same richer environment provided by single-user applications is not a good solution as users will not likely be willing to learn and use a new application just because of some less frequently-used functionalities [15]. On the other hand, modifying existent single-user applications to support cooperative features may not be possible, not because of the implementation efforts, but because most of them are closed-source. Additionally, separating single-user functionalities from cooperative issues may be advantageous.

The solution proposed in [49], called *transparent adaptation*, is to provide applications with cooperative abilities without changing their source-code. The goal is to have multiple users work on their normal single-user applications in an *unconstrained cooperation*, which means that users access data in a concurrent and freely way. Ensuring that users can be performing changes in the same region or adjacent areas, without using complex merging schemes, is resolved by the *Operational Transformational* technique.

To implement the transparent adaptation approach, below the single-user applications two new middleware layers must be created: (1) the *Operational Transformation Layer* to provide eventual consistency guarantees; (2) the *Adaptation Layer* to translate the linear operations performed in the application into simple primitive operations perceivable by the OT Layer.

To increase system modularity, reduce design complexity and promote component reusability, the application specific functionalities and the cooperation capabilities were completely separated. Indeed, the OT Layer is generic and application independent. This way, only the Adaptation Layer must be written in accordance to the single-user application; the bottom-most layer is the same for every application.

The transparent adaptation technique was successfully used to create *CoWord* and *CoPowerPoint*, the cooperative versions of MS Word and MS Power Point. However, this solution can only work in applications with a suitable API which can be used to intercept and replay user input events, and whose data and operational models are adaptable to the underlying OT technique. If not, implementing the Adaptation Layer either will not be so straightforward or it will even be impossible. Additionally, this solution does not provide any kind of awareness and so users may not be able to easily coordinate themselves.

The FEW file system

The *Files EveryWhere* [4] is a distributed file system for mobile computing that provides high availability, good performance and low energy consumption. It works with a generic *operational transformation* reconciliation strategy combined with an *epidemic dissemination algorithm*. This means that users will work *asynchronously* and *concurrently* without the need to perform any conflict resolution task. On the other side, the reconciliation mechanism must be adapted to the specifics of each file type.

Updates in the FEW-FS are propagated using first a *best-effort propagation* system that accelerates convergence and then, to guarantee eventual convergence between replicas, they perform *periodic pair wise epidemic propagation sessions*. In these sessions, replicas send and receive al the unseen updates regardless of their origin.

To merge concurrent updates from different users in order to achieve a common final state in every replica, FEW first translate updates into semantically-rich operations. To do this translation it uses a type-specific program, which compares the previous and the new version of the file. Afterwards, the inferred set of operations is propagated to the remaining replica (using either of the previously mentioned strategies). When an operation arrives to a certain replica, it is integrated employing the generic operational transformation algorithm (the GOTO [38] algorithm in this case).

The *reconciliation method* presented in [4] is suitable only for text files. However, the solution can be adapted to other types of files. This approach, allows maintaining multiple versions for each line as users modify them concurrently, much like the CVS approach. Yet, it allows the merging of multiple versions to be postponed and so to continue the modifications even if they include modifying the lines with multiple versions. This way, there are no lost updates and user intentions are preserved since operations are not automatically merged.

Additionally to the presented method, the authors propose also two solutions to deal with files regardless of their content. The first consists in choosing one of the versions for all replicas. The second solution will keep all versions originated by concurrent updates so that the user can later access and eventually merge them.

2.2.2 Distributed Wikis

The Wooki

The increase popularity and importance of *wiki systems* in the daily life of Internet users and mainly of institutions (academic, research, non-governmental organizations, etc.) and corporations unfolds a series of critical issues related to availability. In fact, nowadays, most popular wikis are designed in a purely centralized infrastructure. This means that availability can be compromised in case of *failure*, *heavy load* or *offline access*. If, on the other hand, wikis were replicated in a *P2P network* there would not be any problems with *fault tolerance* or *load balancing*. Additionally, even the *hardware costs* could be divided between all the peers.

As the solution of replicating wikis effectively handles the problems of the centralized version, it arises the problem of *maintaining consistency*. The *Woot* [27], overview in section 2.1.4, algorithm can be used to deal with this consistency problem. This algorithm is improved and incorporated in a fully functional P2P wiki system called *Wooki* [46]. Additionally, to solve the update propagation problem which is not covered in [27], the wooki system uses a *probabilistic dissemination algorithm* combined with an *anti-entropy algorithm* for managing failures or disconnected sites. Also introduced with *wooki* is an improved version of the woot algorithm called *Wooto* [46], which achieves better performance and has smaller memory requirements.

In wooki, as usually in wiki pages, users make all their changes to the page and only then save them. Subsequently, a diff algorithm is used to translate these modifications in wooto operations (using the *delete* and *insert* primitives). Then, operations are immediately executed locally and broadcasted using the *lightweight probabilistic broadcast* (*lpbcast*) [12]. This algorithm ensures dissemination of messages to all connected nodes. To deal with offline operations, an anti-entropy algorithm was used. This means that sites periodically send missing updates to randomly selected neighbours. Sometimes, if a site stays offline too long, anti-entropy recovery might not be possible. To solve this situation, wooki sites can update themselves from their neighbours in a state-transfer manner.

DistriWiki

As said before, *Wikis* are growing in importance over the Internet. They are one of the best solutions for cooperative publishing [22]. However, the read-only approach of Web sites that dominates the Internet lead to the establishment of Wikis as centralized services, which does not articulate well with their distributed nature.

In [22], the authors propose that Wikis, in order to comply with their distributed usage paradigm, should be implemented over a peer-to-peer infrastructure. The presented prototype, a peer-to-peer wiki called *DistriWiki* provides the same set of functionalities as classic wiki systems but without the reliability problems of centralized solutions.

The work of DistriWiki identified five important features of peer-to-peer wikis: *redundant decentralization*; *unique identification of documents*; *efficient retrieval of documents*; *usability*; *compatibility with wiki concepts.* This means that P2P wikis should distribute data across peers, which should be able to freely change the content of the distributed data. Also, users should still be able to search and retrieve documents independently from the failure of any peer. Finally, the tools should have a similar user interface and similar concepts and semantics as current wiki systems.

One problem with the completely decentralized solution is how to handle conflicts motivated by concurrent changes. Unfortunately, automatic conflict handling was not covered by DistriWiki, which simply leaves for users to handle such conflicts.

UniWiki

Current peer-to-peer systems are mostly used for distributing quasi-immutable content, thus providing high data availability and good performance. On the other hand there are little solutions for ensuring both *scalability* and *consistency* in the case of highly dynamic content. The architecture presented in [26] is suitable for systems that need to *store huge amounts of data*, make use of *P2P networks*, does not have *any single point of failure* and can handle *concurrent updates* ensuring *eventual consistency.*

This architecture was used to create a peer-to-peer wiki called *UniWiki* (Universal Wiki), which relies on *distributed hash tables* (DHT) and *optimistic replication* [34]. The normal structure of DHTs, which were created for storing actual data, was slightly modified to *store operations* in each node so it could fit the imposed consistency requirement. To ensure eventual convergence of data, like in *Wooky* [46], the *Woot* [27] algorithm was used. In order to avoid having every node storing the entire set of wiki pages, each of them is responsible for only a well defined chunk of the content. The defined limits of these chunks can change dynamically in case of arrival or departure of nodes. Another important architectural detail is the separation of the DHT storage system from the wiki clients. To allow this separation, wiki *front-ends* handle client requests and either transform them in Woot operations (update request) or retrieve the wiki content rendered into HTML (page view request). This allows the system to be *transparently integrated* with other existing wiki engines (e.g. *MediaWiki*[1]). To support the integration, the system intercepts wiki calls using MediaWiki so that the distributed storage system is used. The extensive description of the interceptors can be seen in [26].

2.2.3 Intra and Inter-document Relationships

The provide *locality-awareness* in *Cooperative Work Tools*, whether in a *document editor*, *wiki*, *spreadsheet editor*, *presentations builder*, etc., it is of the most importance to know the semantic organization of the data under edition. For example, if one user is editing a chapter, he will want to know if another user starts editing that same chapter. One will be even more interested in knowing if another user starts editing the same section one is editing and even more if it is the same paragraph.

However, this concept of distance between users working concurrently based on the semantic structure of the data being edited must be enhanced if we consider some types of files. For example, it is normal in a HTML or Wiki page to have links to regions inside the same or other pages. Also, in spreadsheets formulas almost always refer to cells placed in the same or other sheets or even in another file.

[1]http://www.mediawiki.org

This intra and inter-document references can shorten the natural semantic distance between regions and systems that aim to provide *locality-awareness* must consider them depending or their context.

There is already some work in this field. In the wiki environment, there are several systems, called *semantic wikis* [6, 45, 19, 29], which aim to subtract from wiki pages relationships between data within pages and among pages. The objectives of semantic wikis are however different since they aim for the exportation of the wiki-based knowledge to databases, ontologies, etc. Another example of reference management is the treatment of *dangling references in the web* [17, 21, 43], in which links are considered to ensure referential integrity in the web.

Chapter 3

Architecture

In a *cooperative system* which applies a *strict or total consistency model*, as new updates are issued by clients, they are immediately sent to the server. Secondly, as they arrive at the server, they must be sent immediately to every client and be applied by every member of the *cooperation group*. This means that every new update has to be either received or sent by the server as many times as the number of users. This total consistency approach works well for systems with a reduced number of clients and a small rate of creation of new updates. But as the rate at which updates are generated and the number of clients escalate, servers in these systems become a serious bottleneck to the overall system performance since they get overloaded with requests and the time required to respond to them increases.

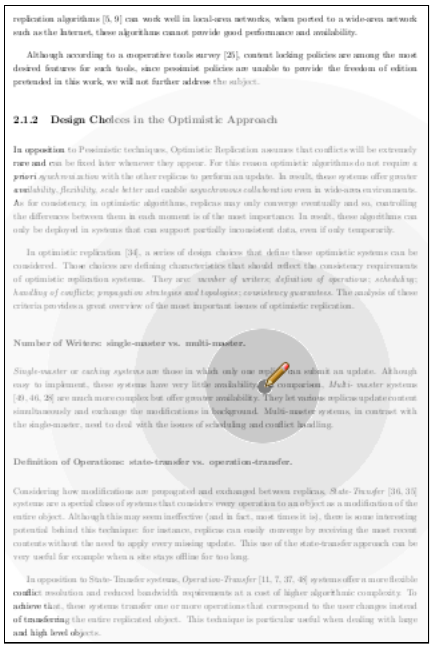
The solution to this bottleneck problem is based on the decision of which updates must be immediately sent and which can be stored for a while and only later be sent. If we can make this decision, we will be able to combine the temporarily stored updates into lesser and overall smaller ones. Consequentially, we will reduce the usage of network resources and prevent the server form becoming flooded with requests which it cannot respond to in an acceptable time.

In this chapter we start by defining the VFC model in the new environment of Cooperative Work, namely in *Cooperative Editing of Documents*. Following, in section 3.2, we describe the architecture of a VFC powered system. Finally, in section 3.3, we explain in detail how to enforce the new VFC model.

3.1 VFC for Cooperative Work

As seen in section 2.1.5, VFC (Vector-Field Consistency) is a Continuous Consistency Model (section 2.1.5) in which location aware techniques are used to balance consistency between replicas in a distributed system. Thus, using information about each *user location* in the replicated object space, the model imposes different consistency limits to each one of them. VFC was initially designed to meet the requirements of a distributed ad-hoc gaming scenario. In this section we describe how VFC can fit in the environment of cooperative work, namely the cooperative edition of documents.

The great interest about this adaptation is to see how we can attempt to infer which contents are more relevant to the user, or, in this case, the editor, based on its position in the *structure of the distributed contents* (for example, in the *chapter* V, *section* Integration, *subsection* Implementation Issues).



replication algorithms [5, 9] can work well in local-area networks, when ported to a wide-area network such as the Internet, these algorithms cannot provide good performance and availability.

Although according to a cooperative tools survey [25], content locking policies are among the most desired features for such tools, since pessimist policies are unable to provide the freedom of edition pretended in this work, we will not further address the subject.

2.1.2 Design Choices in the Optimistic Approach

In opposition to Pessimistic techniques, Optimistic Replication assumes that conflicts will be extremely rare and can be fixed later whenever they appear. For this reason optimistic algorithms do not require a priori *synchronization* with the other replicas to perform an update. In result, these systems offer greater availability, *flexibility*, *scale better* and enable *asynchronous collaboration* even in wide-area environments. As for consistency, in optimistic algorithms, replicas may only converge eventually and so, controlling the differences between them in each moment is of the most importance. In result, these algorithms can only be deployed in systems that can support partially inconsistent data, even if only temporarily.

In optimistic replication [34], a series of design choices that define these optimistic systems can be considered. Those choices are defining characteristics that should reflect the consistency requirements of optimistic replication systems. They are: *number of writers*; *definition of operations*; *scheduling*; *handling of conflicts*; *propagation strategies and topologies*; *consistency guarantees*. The analysis of these criteria provides a great overview of the most important issues of optimistic replication.

Number of Writers: single-master vs. multi-master.

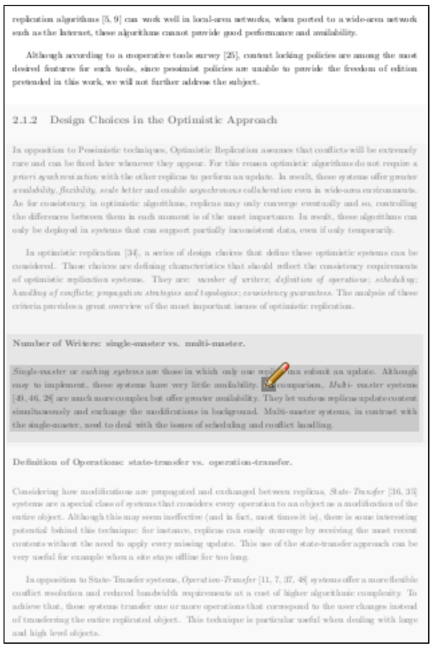
Single-master or caching systems are those in which only one replica can submit an update. Although easy to implement, these systems have very little availability. In comparison, *Multi- master systems* [49, 46, 28] are much more complex but offer greater availability. They let various replicas update content simultaneously and exchange the modifications in background. Multi-master systems, in contrast with the single-master, need to deal with the issues of scheduling and conflict handling.

Definition of Operations: state-transfer vs. operation-transfer.

Considering how modifications are propagated and exchanged between replicas, *State-Transfer* [36, 35] systems are a special class of systems that considers every operation to an object as a modification of the entire object. Although this may seem ineffective (and in fact, most times it is), there is some interesting potential behind this technique: for instance, replicas can easily converge by receiving the most recent contents without the need to apply every missing update. This use of the state-transfer approach can be very useful for example when a site stays offline for too long.

In opposition to State-Transfer systems, *Operation-Transfer* [11, 7, 37, 48] systems offer a more flexible conflict resolution and reduced bandwidth requirements at a cost of higher algorithmic complexity. To achieve that, these systems transfer one or more operations that correspond to the user changes instead of transferring the entire replicated object. This technique is particular useful when dealing with large and high level objects.

Figure 3.1: Original VFC Consistency Zones



replication algorithms [5, 9] can work well in local-area networks, when ported to a wide-area network such as the Internet, these algorithms cannot provide good performance and availability.

Although according to a cooperative tools survey [25], content locking policies are among the most desired features for such tools, since pessimist policies are unable to provide the freedom of edition pretended in this work, we will not further address the subject.

2.1.2 Design Choices in the Optimistic Approach

In opposition to Pessimistic techniques, Optimistic Replication assumes that conflicts will be extremely rare and can be fixed later whenever they appear. For this reason optimistic algorithms do not require a priori *synchronization* with the other replicas to perform an update. In result, these systems offer greater availability, *flexibility*, *scale better* and enable *asynchronous collaboration* even in wide-area environments. As for consistency, in optimistic algorithms, replicas may only converge eventually and so, controlling the differences between them in each moment is of the most importance. In result, these algorithms can only be deployed in systems that can support partially inconsistent data, even if only temporarily.

In optimistic replication [34], a series of design choices that define these optimistic systems can be considered. Those choices are defining characteristics that should reflect the consistency requirements of optimistic replication systems. They are: *number of writers*; *definition of operations*; *scheduling*; *handling of conflicts*; *propagation strategies and topologies*; *consistency guarantees*. The analysis of these criteria provides a great overview of the most important issues of optimistic replication.

Number of Writers: single-master vs. multi-master.

Single-master or caching systems are those in which only one replica can submit an update. Although easy to implement, these systems have very little availability. In comparison, *Multi- master systems* [49, 46, 28] are much more complex but offer greater availability. They let various replicas update content simultaneously and exchange the modifications in background. Multi-master systems, in contrast with the single-master, need to deal with the issues of scheduling and conflict handling.

Definition of Operations: state-transfer vs. operation-transfer.

Considering how modifications are propagated and exchanged between replicas, *State-Transfer* [36, 35] systems are a special class of systems that considers every operation to an object as a modification of the entire object. Although this may seem ineffective (and in fact, most times it is), there is some interesting potential behind this technique: for instance, replicas can easily converge by receiving the most recent contents without the need to apply every missing update. This use of the state-transfer approach can be very useful for example when a site stays offline for too long.

In opposition to State-Transfer systems, *Operation-Transfer* [11, 7, 37, 48] systems offer a more flexible conflict resolution and reduced bandwidth requirements at a cost of higher algorithmic complexity. To achieve that, these systems transfer one or more operations that correspond to the user changes instead of transferring the entire replicated object. This technique is particular useful when dealing with large and high level objects.

Figure 3.2: Adapted VFC Consistency Zones

To adapt the VFC model, which was initially created for a totally different environment, its main concepts must be adjusted, namely *replicated objects*, *pivots*, *consistency zones* and *distance*. Also it is not entirely straightforward how to determine to which objects, *consistency vectors* will be applied.

3.1.1 Adapting VFC Concepts

In the VFC Model we define the state of the replicated world as the aggregate of states of all the objects present in that world. For example, in a RPG, each avatar presented in the game, whether computer or user controlled, is one of these *replicated objects*. In the scene of cooperative edition of documents, the state of the world, or in other words, the state of the document will be given by the sum of the states of every of its divisions. These divisions can go from entire regions (chapters, sections, paragraphs, etc.) to simple lines or even characters. For architectural reasons which will be further explained in section 3.2.3 we defined such objects (replicated units) as single characters.

Another concept of the most importance is the concept of *pivots*. They roughly correspond to each user's observation point. In this new environment, instead of having a coordinate based definition of the pivot position, which would possibly be defined here by the character number, since that is the base for inferring the content relevance, we define the pivot position by its place in the structure of the document. Additionally to the position of the user in the document, there can be several other user defined points of interest present in the user's set of pivots.

By having a different set of pivots for each user, changes in the document will affect each user differently. The same is to say that consistency will vary from user to user. This variance is defined by the *distance* between user pivots and the various replicated objects. In more practical terms, consistency between the contents in a semantic region of the document should weaken as the distance to the user pivot grows. Thus, distance is defined as the distance between the semantic regions being edited and the closest pivots.

To enforce these different consistency levels, around the pivots of each user we define several *consistency zones*. Consistency zones represent zones in the document in which replicated objects are at a well defined distance to the user pivots and have a certain level of importance to the user's own editions. The most important zones are those immediately surrounding the user pivots which contain objects at distance 0 and so where consistency is stronger. As we go trough the remaining zones at greater distances, relevance decreases and consequentially consistency weakens. In the original VFC, these consistency zones were defined as concentric ring-shaped areas around the user pivots (Fig. 3.1). Naturally, and to fit the objectives of the adaptation, this definition cannot be directly applied to the edition of documents. Thus, consistency zones must be defined according to the document structure. Hence, we can determine which structural divisions of the document are closer to each other and which are farther. In Figure 3.2 there is an example of how consistency zones could look like around a certain point of edition in a document.

Another defining difference between the original VFC model and its adaptation to the cooperative work scenario is that, in the later, the boundaries of the replicated contents change. Meaning that in the gaming scenario the game map coordinates were immutable. But in the cooperative work scenario, updates can change the very structure of the document (for example, when we add or remove a section to a document) which is the basis for the locality-awareness in the VFC model. These changes might affect the user's position and consequentially the distances and the consistency zones. In conclusion, other than keep tracking of the user's position and the arrival of new updates, we also have to keep track of changes in the structure of the replicated document.

In the VFC model, for each consistency zone, we define a *consistency vector* to enforce *divergency limits* for objects inside the zone. The 3-dimensional consistency vector [θ, σ, ν] bounds the maximum divergency for a replicated object respectively in terms of *time* (θ, maximum time without updates), *sequence* (σ, maximum number of unseen updates) and *value* (ν, maximum difference between the user and most up-to-date versions). In the original model, the limits were enforced individually for each replicated object. However, if we ported this technique directly to the new environment, we would be bounding sequence and value limits for each character (the smallest replicated unit in this work), which is not possible since a character cannot change its value, i.e. is immutable. Therefore, and in accordance with to the objectives of this adaptation, this limits have to be enforced on a per-region basis.

As it can be perceived from these definitions, the adaptation of VFC will greatly depend on the type of documents being edited. For example, the precision of the user location and the distance between objects are directly dependent of the granularity of the document structure. Also, the consistency zones change depending on which types of structural regions we have defined for the document. Next, we will see in more detail how these concepts were adapted to a Latex document.

3.1.2 Adapting VFC Concepts to a LaTeX Document

In this section we explain in detail how the VFC concepts were adapted to a data space ruled by the structure of a Latex Document, the chosen type of document for this work. Differences to other types of documents rely in the type of divisions available and distances between them. However, because of the close relation between the definition of distance and consistency zones, the latter will also be affected. This means that the only concept that is independent of the type of document is the user's position which only requires that there is a somewhat standard representation of the document structure. Nonetheless, analogous adaptations can be designed to other document formats following a similar structure.

Structure of the Latex Document

There are various ways to obtain the structure of a Latex file. The most precise involves the use of Latex compilers. Although there were some benefits in having such a correct structure, for the purpose of this work, having an almost always correct structure was sufficient. This simplifies development and editor enhancement. Hence, a Latex document parser was created to extract the structure of the document.

The structure of the document extracted by the developed parser is the following:

- *Preamble*: Everything before the beginning of the *Document* region;
- *Document*: Everything between \begin{*document*} and \end{*document*};
- *Ending*: Everything after the end of the *Document* region;

Inside the Document (region), we can have the following regions:

- *Abstract*: Everything between \begin{*abstract*} and \end{*abstract*};
- *Sectioning Command (Main Contents)*: Everything from the beginning of a sectioning command to the beginning of the next. Sectioning commands can be divided (from the outermost to the innermost level) in *Appendix*, *Part*, *Chapter*, *Section*, *Subsection* and *Subsubsection*.

Note that further subdivisions, like paragraphs or even lines, would have been possible. However, since we wanted to apply the VFC model based on the relations between greater regions of the the document we choose not to go deeper into the structure.

Distances between Regions and resultant Consistency Zones

Distances between objects and pivots in a replicated document depend on its structure. What we do is to translate absolute locations in the document content into normalized positions inside the document structure. Thus, we define the distance between two points in the document as the distance between the regions in which they are located. This section will explain how the distance between every pair of regions is calculated. Throughout the explanation, we will be giving examples of the consistency zones around pivots located in the different regions.

The most basic case of a document is one without any kind of structure (for example, a document without the \begin{*document*} command). In this case, since no assumptions can be made about the relations between objects based on their location in the structure, we consider that the distance between every two objects is 0. This is the same as having a totally/strictly consistent replicated document (see Figure 3.3).

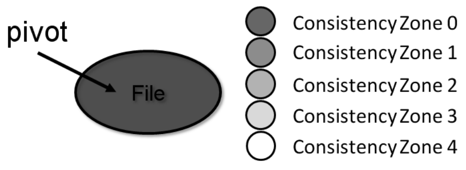
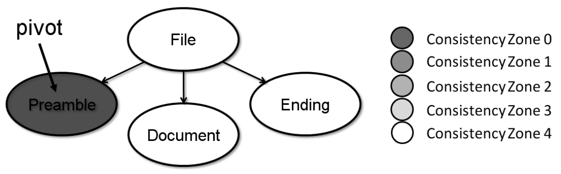
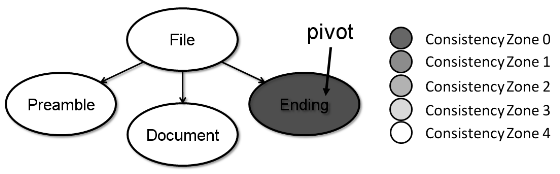


Figure 3.3: Consistency Zones around the File region

When we have some structural divisions in the document, at least, the following top-level regions are present: the *Preamble*, the *Document* and the *Ending*. The Preamble is where we usually find a series of commands that will affect appearance and definitions of the resulting document. In the Document (region), we can find the contents to be generated. Finally, at the Ending, editors may place any unformatted data, like commentaries, listing of chores, etc. The Latex semantics already separate these 3 top-level regions. Indeed, Latex values promote the separation between contents and the way they are presented. On the other hand, the Ending region is simply ignored by the Latex compiler and so their contents cannot be related with any other contents. Taking this in consideration, we say that the distance between an object located in one of these regions and an object located in another one is maximized (see Figure 3.4).



(a) Around the Preamble



(b) Around the Ending

Figure 3.4: Consistency Zones around the top-level region

Inside the Document, the definition of distance is more complex, Contents within this region are divided in the *Document beginning*, the *Abstract* and the *Main Contents* (composed by a series of *Sectioning Commands*). In the Document beginning (everything from \begin{*document*} to the Abstract or the first Section Command), we usually encounter information about the resulting document, like the title or the author. Since these contents do not depend much on the ones in the Abstract or the Main Contents, the distance between the Document beginning and the Abstract or the Main Contents is defined as almost the greatest possible (see Figure 3.5).

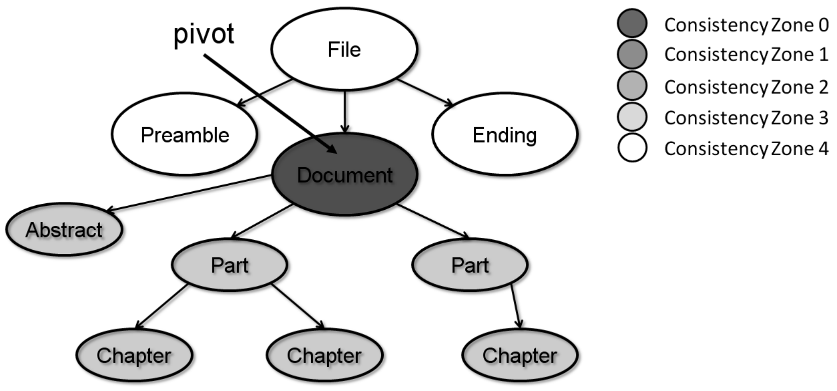


Figure 3.5: Consistency Zones around the Document region

As for Abstract, since it is an overview of the subjects described in the document, the Abstract depends highly on the contents of the outermost regions of the Main Contents. However, as we go further into the contents of the innermost regions, they start to become too specific to conditionate updates in the Abstract. This means that distance between the Abstract and the Main Contents increases as we go down in the structure into deeper regions. In the Figure 3.6 we can see that exact behavior.

The most common edition scenario is to have the users located somewhere inside the Main Contents (group of regions delimited by a sequence of sectioning commands). In this case, we say that the contents

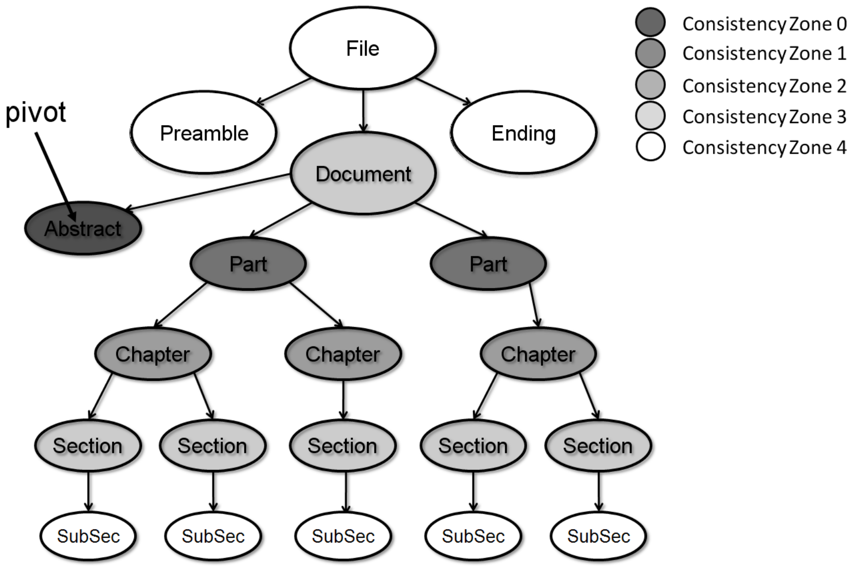


Figure 3.6: Consistency Zones around the Abstract region

of a specific region are highly connected with the contents of the containing regions and also with the contents of its subregions. For example, a specific section is highly related with the containing chapter and, in the opposite direction, with its own subsections. But if we go further away from that section, the dependence between contents gets weaker and consequently the defined distance increases. Additionally to these direct structural relations, a region in the Main Contents is also linked with the adjacent regions. For instance, the contents of a subsection are greatly related with the contents of neighboring subsections. Again, the closer the region is, the smaller the distance is. An interesting detail about this relation is that neighboring innermost regions have stronger content dependence than neighboring outermost regions. If we think about specific examples, we see that two subsections next to each other are strongly connected than two parts or two chapters.

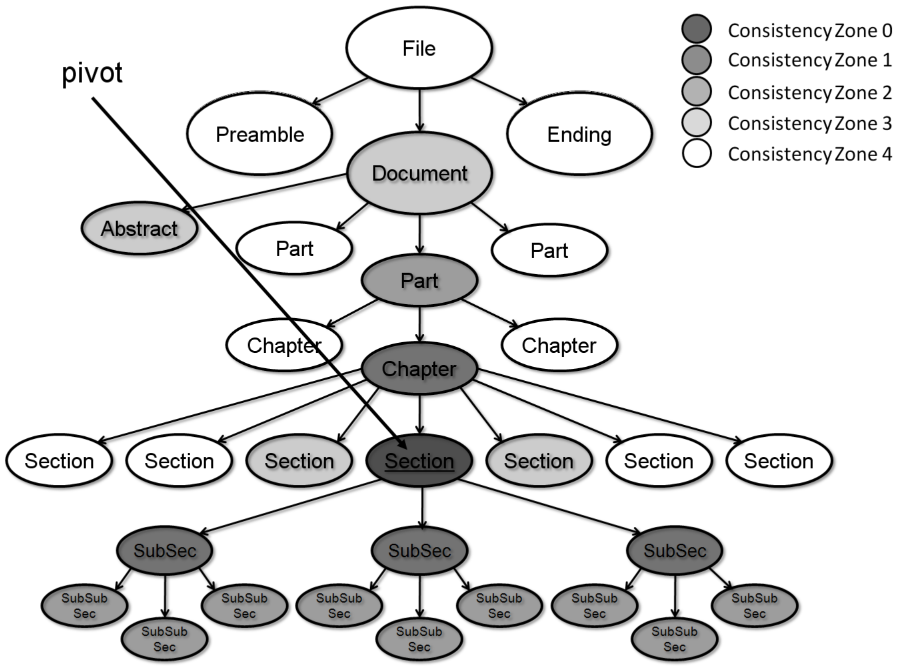


Figure 3.7: Consistency Zones around a Section

Still in the sectioning commands, note that the definition of consistency zones comes closer to the definition in the original VFC environment. This similarity can be seen in Figure 3.8, where, around the pivots in the document structures, we have consistency zones containing almost all the regions at a certain number of steps from the first in the same direction.

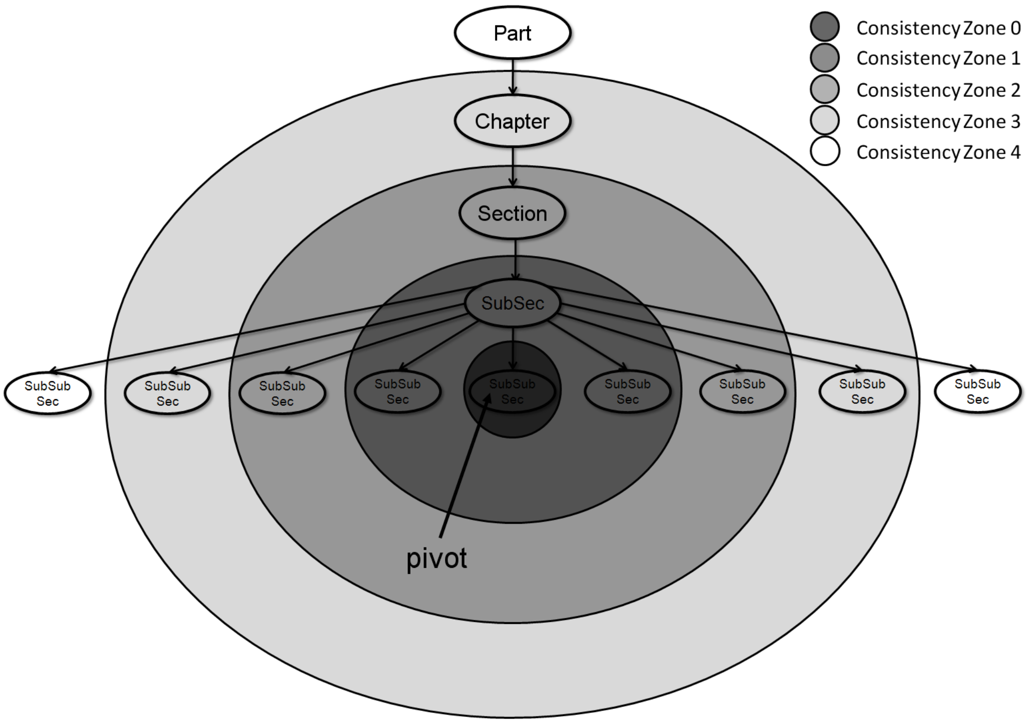


Figure 3.8: Similarity to the original Consistency Zones

Finally, as it may be perceived by the previous examples, when in the same structural region, two objects will be at 0 distance from each other, meaning that contents inside the same region are totally related with each other.

Zone	*Time* (θ)	*Sequence* (σ)	*Value* (ν)
0	1 sec.	1 update	1%
1	10 sec.	15 updates	5%
2	40 sec.	100 updates	30%
3	2 min.	750 updates	60%
4	5 min.	1000 updates	90%

Table 3.1: Consistency Vector values for each Consistency Zone

Regarding the examples just presented, we defined five different consistency zones (distances from 0 to 4). For each one, we defined a specific consistency vector with the time, sequence and value limits (Table 3.1). Each region belonging to these consistency zones is related to the pivot region in a different way. We now give a more intuitive description of those relations that will help define consistency zones in a similar type of structure:

- *Consistency Zone 0*: Regions with the same information (ex: the same *subsubsection*).
- *Consistency Zone 1*: Regions with much in common (ex: adjacent *subsubsections*).
- *Consistency Zone 2*: Regions sufficiently related for a relatively frequent update (ex: containing *section*).

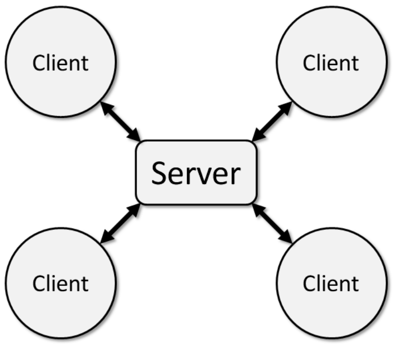


Figure 3.9: Node distribution in the network

- *Consistency Zone 3*: Regions probably unrelated (ex: containing *chapter*).
- *Consistency Zone 4*: Regions certainly unrelated (ex: *preamble*).

3.2 System Architecture

Cooperation using VFC[1] is based on the notion of a *cooperation group*. A cooperation group is a cluster of peers in a network working together to create the illusion of a single copy being edited by a group of users, despite in reality each being editing his own copy of a replicated object. In this work, a cooperation group only manages the updates to a single document.

The system supports concurrent editions in the document regardless of the order in which they happen. This implies the absence of any lost updates due to conflicting updates or any divergence between the local replicas of clients.

Next, we describe the *Network Architecture* of the system. Then, in section 3.2.2 we describe in more detail the different *Software Components* present in the system. Finally, in section 3.2.3 we explain the most relevant data types are represented in system, that is the document and the updates to it.

3.2.1 Network Architecture

The distributed system that supports the cooperation group is composed by an aggregate of client nodes and a single server node, just as described in Figure 3.9. (and following the original VFC architecture). Although this is out of the scope of this work, the system can be easily enhanced to provide tolerance to server faults. For instance, we could introduce system nodes which would act as server backup units and would be strictly synchronized with the server to assume his role in case of failure.

An interesting aspect about this configuration is that all VFC reasoning is confined to the server. This has the great advantage of hiding the VFC behavior from the clients, which highly simplifies the adaptation. Also, this way, we have a node with an updated view of the whole document and consequentially of its structure. This will result in more accurate distance measurements when comparing this solution to one where VFC reasoning is distributed for all clients.

[1]In fact, this is also verifiable when not applying the VFC model.

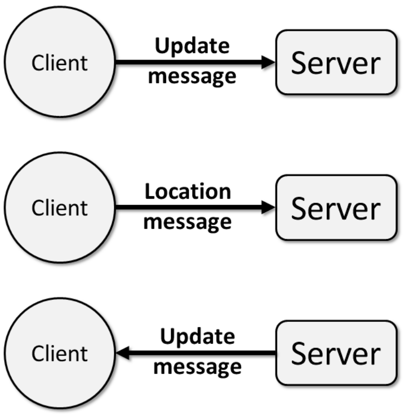


Figure 3.10: Messages exchanged between the Client and Server nodes

Client nodes are responsible for proactively sending their document updates to the server as soon as they happen. Also, since VFC is a location-aware model, clients also have to tell the server their editing location in the document. This may happen in a explicitly way, through a special-purpose message or implicitly, if the server extracts the edition location from document update messages. This duality gives a flexibility in the client implementation since the adapted application framework is not required to provide position change signals.

Server nodes are responsible for the management of entrance and exit of clients in the cooperation group. Additionally, the server has to send document updates to the clients in the group. In a total consistent system, the server acts more like a propagation gateway. But in a VFC powered system, like this one, the server can also decide to store the updates and delay their propagation.

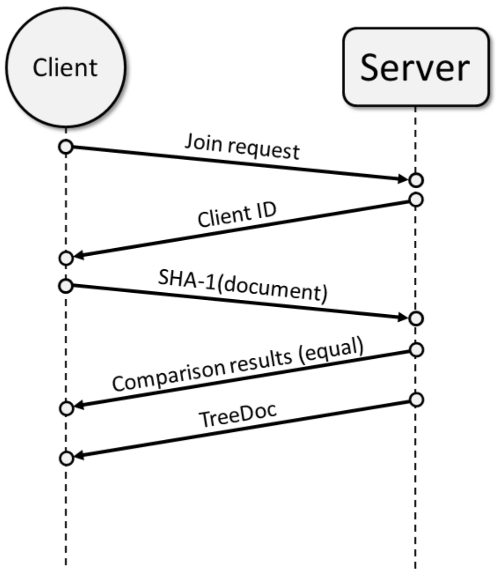
Client and Server Messages

There are two types of messages exchanged between the client and server (see Figure 3.10):

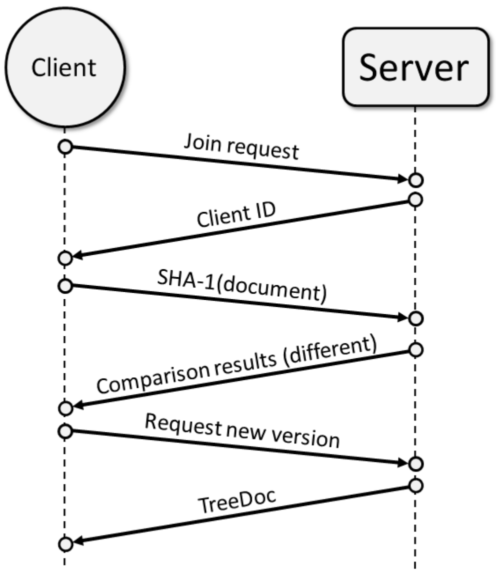
- *Update message*: This message carries updates to either be applied in the local replica (from server to client) or to be propagated to other clients (from client to server). The representation of these updates is dependent of the shared document representation, as will be explained later in this section. When from the client, this message must be immediately sent after an update. When from the server, it depends on the enforcement of VFC limits (*consistency vectors*).
- *Location message*: This is a special-purpose message from the client to the server to inform the latter about the client location. The frequency of this message is not defined and depends on the client application framework and adaptation: it can be in fact a periodical message, but it can also be sent whenever the client application finds necessary, and finally it can even not be sent at all.

Client Entrance Protocol

Once the server is running, users who wish to join the cooperation group must connect to the server. Since the server is totally separate from the client, even the user who launched it has to explicitly connect to it. When a request to join the group reaches the server, the negotiation goes as follows (Figure 3.11):



(a) Client and Server documents are equal



(b) Client and Server documents differ

Figure 3.11: Protocol for Client entrance in the distribution group

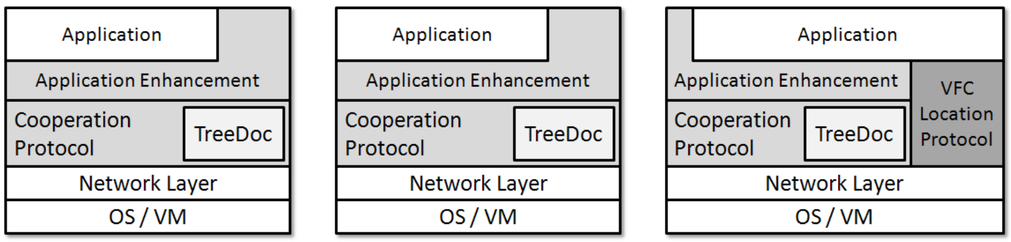
- Client opens a connection with the server and sends a request to joint the group.
- Server sends the client-id which will later be used as a *tree disambiguator* (section 2.1.4).
- Client uses the SHA-1 [10] hash function (or another type of unique identifier generator) to process the document and generate an unique identifier which is sent to the server.
- The server compares the client and server contents and send the comparison results back to the user.
- If the documents differs, after asking permission the user, the client request the most recent version to the server.
- If the documents are similar or the server receives a request for the most recent version, it sends the TreeDoc representation of the document to the client (TreeDoc will be explained later in this section).

After this process, the Client has entered in the cooperation group and is now able to send and receive document updates to and from the server.

3.2.2 Software Components

Client

Client nodes correspond to the adapted editor applications run by users. They asynchronously edit the local replicas of the shared document and only afterwards send updates to the server node. Furthermore, the client monitors the user activities and listens to any server update messages. As soon as the updates arrive, they are immediately applied.



(a) Total Consistency version

(b) VFC version without explict location messages

(c) VFC version with explict location messages

Figure 3.12: Client adaptation Layered Architecture

An interesting aspect of the VFC adaptation is that, as far as the client nodes know, there is no special technique filtering which and when each client should receive the new updates. Because of this unawareness of the VFC model, client applications suffer very little or no changes to be able to adapt to the VFC mode. As seen in Figure 3.12, if we choose not to use explicit location messages, the client architecture remains the same (Figure 3.12(b)). If on the other hand, we have implicit and explicit location messages we may have to adapt the client (Figure 3.12(c)). Since we were working with an open source application and we wanted to have a complete VFC enabled scenario, in this work we have both implicit and explicit location messages.

Additionally to the core application, a complete VFC Client (Figure 3.12(c)) has three fundamental layers: *Application Enhancement*, *Cooperation Protocol* and *VFC Location Protocol* layers. Next we describe the three in more detail.

Application Enhancement Layer This layer, which matches application semantics, is responsible for keeping track of any update to the application copy of the document. Also, it translates more complex operations into simple *add* and *remove* operations, which can be passed to the *Cooperation Protocol* layer. Additionally to the monitoring of modifications to the document, this layer receives updates from the Cooperation Layer and updates the application copy of the document, which is held by the Application layer. Finally, this layer also interacts directly with the user, generally to warn him about error messages.

Cooperation Protocol Layer The Cooperation Protocol layer is responsible for managing incoming and outgoing updates. To do it and still enable the user to asynchronously perform editions to the document, this layer holds a TreeDoc representation of the document (a special-purpose structure that supports free concurrent edition without conflicts). This means that the client node has two representations of the document, a plain text representation in the Application Layer and the TreeDoc representation in this layer. To cope with this redundancy, the Cooperation Protocol and the Application Enhancement layers have to work together to guarantee copy consistency.

VFC Location Protocol Layer The VFC Location Protocol layer keeps track of user location in the document. Then, periodically it sends this information to the server. Also, if the difference between the user position and the last position sent to the server changes above a configurable threshold, a location message is sent to the server. If there is no explicit location messages or the Cooperation Protocol layer already provides this functionality, there is no need for this extra layer.

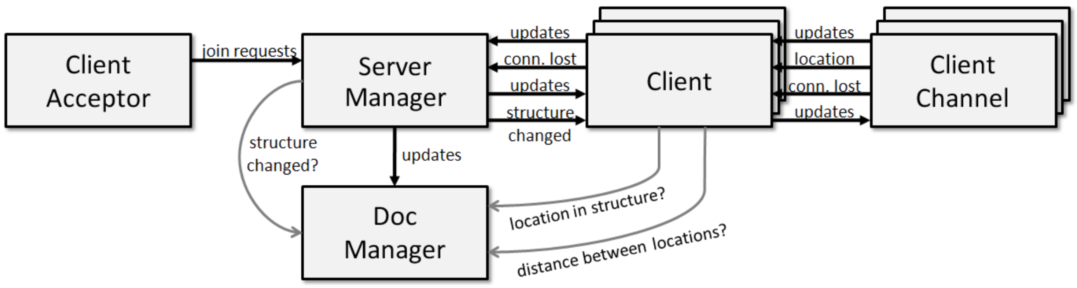


Figure 3.13: Server node components

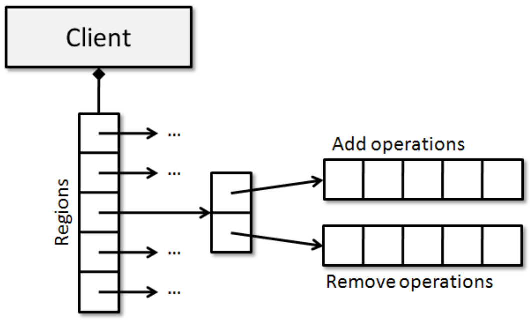


Figure 3.14: Storage of updates (operations) in the Client component

Server

The Server is the central node in the system. Every message is either sent or received by this node. It accumulates client management and update propagation responsibilities. In a VFC powered system it is up to the server to reason about the discriminated delay of updates.

In this node we find five main components (Figure 3.13), each with a well defined function: *Server Manager*, *Client Acceptor*, *Document Manager*, *Client* and *Client Channel*. Next we describe each of these components in detail.

Server Manager The Server Manager is the central component of the Server node. It is responsible for all cooperation group management functionality and, consequentially, has to be aware of group entrance requests and loss of connection from any client node. Additionally, the Server Manager component is responsible for distributing updates to each Client component and for warning them about structural changes in the document.

Client Acceptor This component is responsible for listening to incoming client join requests and send them to the Server Manager.

Document Manager The Document Manager holds the document (in the form of a TreeDoc) and its structure. It is responsible for controlling concurrency in the access to both. Also, this component responds to enquires about structure changes, distance measurements and locations in the document structure.

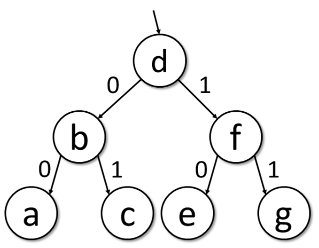


Figure 3.15: TreeDoc representation of the string "abcdefg"

Client There is a Client component for every member of the cooperation group. Each one is responsible for enforcing VFC limits for the correspondent member. Thus, Client holds all the undelivered updates organized as seen in Figure 3.14 (the update semantics will be addressed later in this section) and controls the user location in the document structure. Finally, the Client is responsible for warning the Server Manager about incoming updates. To enforce VFC limits, this component has to be aware of a series of important external events (e.g. *arrival of new operations*, *document structure changes* and *user location changes*). In section 3.3, we will explain how to respond to these events.

Client Channel This component communicates only with the Client (component) and has the responsibility for controlling inbound and outbound communications with the Client (system node) according to the already specified protocol.

3.2.3 Data Representation

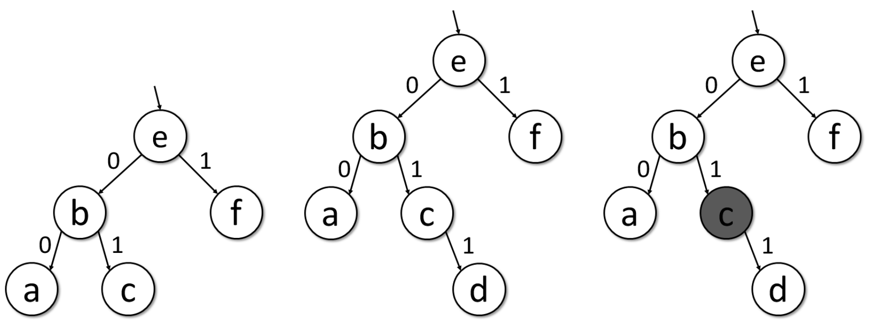
The VFC model requires a great flexibility in the scheduling of updates. On the other hand, since updates might be reordered, there is an increased risk of finding update conflicts. To deal with these issues, the replicated document is a *Commutative Replicated Data Type* (section 2.1.4). This way, we avoid the use of any complex concurrency control providing at the same time the freedom of edition coveted for this work. In particular, all replicas in the system hold the document in form of a *TreeDoc* [37, 28].

Next we will see how documents are represented by a TreeDoc. Then, we explain how the use of a TreeDoc representation affects the representation and encoding of updates. Finally, we present two optimizations to the original definition.

TreeDoc Representation

In order to highly simplify the maintenance of consistency between the client replicas, the document is represented in the form of a CRDT (section 2.1.4). The use of the CRDT, enables replicas to exchange operations freely without the need of any type of conflict resolving. Because of the already explained advantages (section 2.1.4), in this work we used the TreeDoc [37, 28] representation which supplies a unique identifier for each character in the document using an enhanced binary tree (Figure 3.15). Given a tree in this format, the document can be extracted by walking the tree in infix order.

Additionally to the basic definition of the TreeDoc representation, in this work we performed an optimization that results in a compact representation of the TreeDoc which we call a *Partially Expanded TreeDoc*. We also propose an optimization to solve the *Tail problem* which was used in this work only to merge operations. Both these optimizations will be further explained in this section.



(a) A TreeDoc representation of a document

(b) Add('d', 01)

(c) Remove(01)

Figure 3.16: Representation of Updates/Operations

Representation of Updates

The use of a TreeDoc representation of the shared document, implies that updates to the document follow the semantics of TreeDoc updates. As seen in section 2.1.4, this solution is based of an operation-transfer scheme, which indicates that updates are represented by operations, in particular operations to add/remove nodes in the binary tree.

In the Figure 3.16 we can see the representation of operations in the TreeDoc, which can be sent as is to every user. In Figure 3.16(b), we see the results of an operation in the form *Add(character, path)* which represents an operation described as an *insertion of the character 'd' after the character 'c'*. In 3.16(c), we see the results of an operation in the form *Remove(path)* which represents an operation describe as an *removal of the character 'c'*.

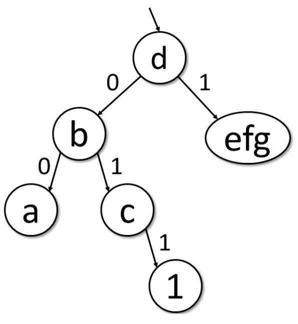
In addition to these basic operation representations, we introduce four others, which will be overviewed when we explain the two optimizations to the TreeDoc representation. They are:

- *Add(string, path)*: insertion of a string of characters;
- *Remove(size, path)*: removal of several characters;
- *Add-tail(string, path)*: insertion of a string of characters in tail form.
- *Remove-tail(size, path)*: removal of several characters in tail form.

The use of an operation-transfer scheme has its problems. One of them is that, in opposition to state-transfer, usually updates cannot be skipped. For example, in a state-transfer system, even if an object is subjected to several updates, the final state is all that matters. On the other hand, in an operation-transfer system, we need to apply all updates in order to determine the final state. However in the particular case of a CRDT, since all operations can commute, if two operations cancel each other, we can simply disregard both. This will create some inconsistencies in the specific TreeDoc of each user, but not in what it represents, the document.

One of the most important aspects of the VFC enforcement in an operation-transfer system is the *Operation-Log Compaction*. When delayed, operations go to an operation log. Since, as explained before,

(a) Non-expanded node holding "abcdefg"



(b) After insertion of '1' between 'c' and 'd'

Figure 3.17: Partially Expanded TreeDoc representation

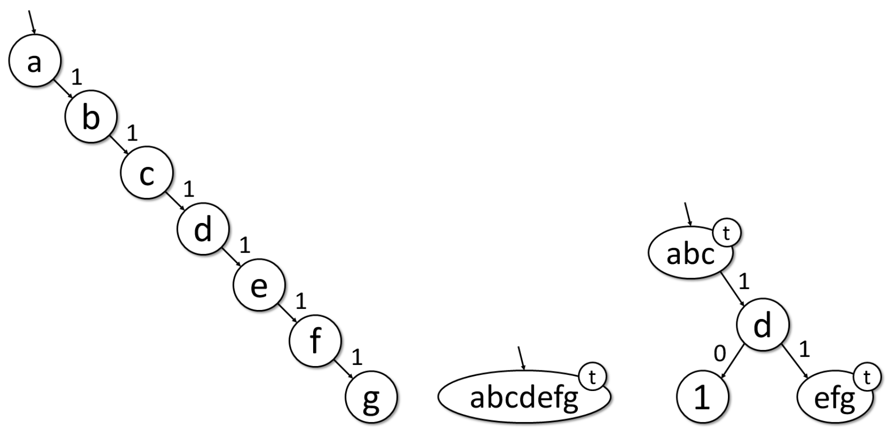
operation-transfer systems do not have automatic *savings in used bandwidth* when updates are delayed, savings must come from operation merging. There are two possible types of operation merging. The first, which was just presented, is the merge between operations that cancel each other. This is only verified when we have an add and a remove respectively inserting and removing the same number of characters in the same position IDs (tree paths). As said, this will result in the elimination of both operations from the log. The second type is a tail-merge which merges several operations into one more compact operation. The technique used to perform this type of merge will be described in this section when we explain the Tail Problem.

Partially Expanded TreeDoc

The TreeDoc original definition requires a node in the document tree for every character (Figure 3.15). Consequentially, as the document size grows, generating the complete tree and sending it to every new client can become tremendous time and space consuming chores. To avoid this problem, we implemented a *partially expanded TreeDoc* representation. In [28], the authors proposed a solution to the structure overheads, which consisted in the application of "*structural clean-up operations*" where they would switch between more efficient sequential buffer representations and edit-oriented representations where there was no node compression. Our solution is inspired by their proposal.

The original idea was to periodically introduce operations that would result in the same representation compressions as in our solution. What we propose is to use a compact version from the very moment of creation of the tree nodes and for as long and as compact as possible. Also, and as observed by the authors in [28], the clean-up operations did not automatically commute, so a distributed commitment protocol would be necessary. The consensus is needed because these operations require that every tree replica is in the same state, i.e. represent the same document. Otherwise, these operations will compromise consistency. Consequently, applying this commitment is problematic in intensive editing scenarios like this where we aim only for eventual consistency (actually divergence bounding, a stricter criterion) and so, replicas are almost permanently in different states. Thus, in order to benefit from this type of efficient representations, compressions cannot just rely on the periodic introduction of clean-up operations.

A partially expanded TreeDoc representation is one where we have non-expanded nodes holding the contents of a certain number of (normal) nodes in a plain string representation (see Figure 3.17(a)).



(a) Result of consecutive insertions (b) Tail-formed node (c) After insertion of '1' between 'c' and 'd'

Figure 3.18: The Tail Problem

Then, only if required (for example when the top node is deleted, or the children are required), that node will be expanded. However, even then, the node's children might still be represented efficiently using this technique as seen in the example of the Figure 3.17(b).

One other advantage of using this compact representation is that it enables entire operations to be represented as simple and smaller ones. For example, when inserting a set of characters all at once, we can represent this operation (and send it as is to other nodes) as an insertion of a non-expanded node. As for the removal of characters, if the nodes represented by a non-expanded node are removed all at once, instead of an operation that expands all nodes and lefts a tombstone in each one, we create an operation that leaves a non-expanded tombstone that represents them all.

The Tail Problem

A serious problem that happens after sometime of editing the document without rebalancing the tree (with a *flatten & explode* operation), is one that comes from the fact that usually, users producing text, do it by writing the characters one by one. This insertion pattern causes what we called in this work as the *Tail Problem*, which is the creation of a great number of nodes with only the right child (Figure 3.18(a)), or as we call it in *tail form*. The biggest issue with this tail problem is that the maximum depth of the tree increases much faster than it should which will influence the size of all position IDs (tree paths) in that branch.

To solve this problem, the insertion of nodes, when in these conditions, will cause a contraction to a tail-formed node, as seen in Figure 3.18(b). Then, like in the previous optimization, whenever necessary this tail-formed node will expand in the required position (see Figure 3.18(c)). The automatic contraction also results with the same benefits in the removal of nodes organized in tail form.

This solution can generate great savings in used memory which are very useful when propagating the tree to new clients. These savings happen since, instead of having to send all nodes in tail form one by one, we only send one node with all the contents. Although, this was not implemented directly in the tree, this very same technique was used to merge stalled operations with great success as we show in the

Seconds	θ	1	2	...	10	11	...	40	41	...	120	121	...	300	301
Consistency Zone 0	1	yes	yes	...	yes	yes	...	yes	yes	...	yes	yes	...	yes	yes
Consistency Zone 1	10	no	no	...	yes	no	...	yes	no	...	yes	no	...	yes	no
Consistency Zone 2	40	no	no	...	no	no	...	yes	no	...	yes	no	...	no	no
Consistency Zone 3	120	no	no	...	no	no	...	no	no	...	yes	no	...	no	no
Consistency Zone 4	300	no	no	...	no	no	...	no	no	...	no	no	...	yes	no

Table 3.2: Timeouts with a period of θ

evaluation results (section 5.2).

3.3 VFC Enforcement

Server nodes are responsible for enforcing the VFC model. In particular, it is up to the Client component to guarantee that as soon as the VFC limits of some document region are reached, all the undelivered updates in that region are immediately propagated.

Note that, some events, (e.g. changes in the document structure) can lead to the sudden and unpredictable disrespecting of the VFC divergence limits. However, since there is no way to prevent (or even foresee) these events or their effects, what we intend to guarantee with the designed techniques is that: *the VFC limits are always respected before and after any update.*

In this section we start be describing how we check *VFC Consistency Vector limits* (section 3.3.1):

- Time (θ)
- Sequence (σ)
- Value (ν)

Afterwards, we define and describe how to handle a set of fundamental events in VFC enforcement. VFC enforcement in the document edition environment can be simplified to the *handling of those pivotal events* (section 3.3.2):

- Arrival of a New Operation (Update)
- User Location Changed
- Document Structure Changed
- Consistency Zone Timeout

3.3.1 Checking VFC Limits

Time

The *Time* slot of a VFC consistency vector defines the maximum time without seeing any updates from a region at a determined distance form the pivot. Like in the original work, to enforce the time limits, we

```
NEW_OPERATION(OP)
 1. region := DocManager.getRegion(OP.locationID);
 2. Client.ops[region].queue(OP);
 3. CHECK_SEQUENCE(region);
 4. CHECK_VALUE(region);
```

Figure 3.19: Handling *new operations*

set a mechanism that signals timeouts for each consistency zone with a period defined by the consistency vector time limit (see Table 3.2).

To create these timeout events, in every Client component, we connect a function to a clock signal with a period equal to the minimum unit of measurement, i.e. one second. This function counts the elapsed time and checks, for every consistency zone, if the time for another timeout event has passed.

Sequence

$$\#ops[region] \geqslant Sequence[distance(region, Client.region)]$$

The *Sequence* limit is defined as the maximum number of unseen updates from a certain region. The Sequence limits are checked in consequence of either the *arrival of a new operation*, or a *change in the user region* or a *change in the document structure*. This limit is checked simply by comparing the number of undelivered operations in a certain region with the maximum sequence limit defined in the correspondent consistency vector.

Value

$$\frac{\sum_{i=1}^{\#ops[region]}\{ops[region][i].size\}}{region.size} \geqslant Value[distance(region, Client.region)]$$

The *Value* limit is defined as the maximum difference between the user replica and the most up-to-date version of a region. This limit is checked in the same situations as the Sequence limit. But in this case we compare the ratio between the number of changed characters and the region size with the maximum Value limit defined in the correspondent consistency vector.

3.3.2 VFC Enforcement pivotal Events

Arrival of a New Operation

When a new operation arrives to the Server node, it will eventually reach every Client component in the Server node. When this happens (Figure 3.19), the operation is stored as described in 3.2.2, i.e. the operation is store according to its location in the document structure. Then, since this new operation might have resulted in the surpassing of the VFC limits for the operation region, the *Sequence* and *Value* limits must be checked.

```
USER_LOCATION_CHANGED(ID)
 1. region := DocManager.getRegion(ID);
 2. Client.ID := ID;
 3. if region <> Client.region then
 4.     Client.region := region;
 5. for each r in DocManager.regions() do
 6.     CHECK_SEQUENCE(r);
 7.     CHECK_VALUE(r);
```

Figure 3.20: Handling *changes in the user location*

```
STRUCTURE_CHANGED()
 1. Client.region := DocManager.getRegion(ID);
 2. for each r in DocManager.regions() do
 3.     for each OP in Client.ops[r] do
 4.         region := DocManager.getRegion(OP.locationID);
 5.         temp[r].queue(OP);
 6. Client.ops := temp;
 7. for each r in DocManager.regions() do
 8.     CHECK_SEQUENCE(r);
 9.     CHECK_VALUE(r);
```

Figure 3.21: Handling *changes in the document structure*

User Location Changed

Explicit or implicit location messages may affect distances between the user and the undelivered operations. To create this effect, the new location must result in a region change. When this happens, the distance between the user region and every other region might have changed. This implies that the VFC limits of every region might have been reached. So, other than updating the user location informations, the handling of this event (Figure 3.20) will also check the *Sequence* and *Value* limits for every region with undelivered operations. Note that, although *Time* limits are not immediately checked, they will be in an interval of less than the period of the clock signal (section 3.3.1).

Document Structure Changed

When new operations arrive at the Server Manager, they are immediately applied in the local replica (held by the Document Manager). Then, the Server Manager has to check if the structure was changed by this last operation. If so, every Client component is signaled. The importance of handling this event (Figure 3.21) is that, since undelivered operations are stored on a per-region basis, structure changes imply the reorganization of all these operations. Also, the distances between regions might have been affected. Thus, after a Client component redistributes the operations, *Sequence* and *Value* limits have to be checked for every document region with undelivered operations. Again, *Time* limits will be checked in an interval of less than the period of the clock signal (section 3.3.1).

Consistency Zone Timeout

This is a special event (Figure 3.22), generated internally by every Client component (section 3.3.1), that keeps track of *Time* limits. When the timeout is reached for a specific consistency zone, every operation

```
CONSISTENCY_ZONE_TIMETOUT(zone)
 1. for each r in DocManager.regions() do
 2. if DocManager.distance(r, Client.region) = zone then
 3.     DELIVER_ALL_OPS(r);
```

Figure 3.22: Handling a *Consistency Zone timeout*

in that consistency zone has to be delivered. So, when the signal for that event is triggered, the Client component will have to check which regions with undelivered operations belong to that consistency zone, and deliver those operations.

Chapter 4

Implementation

In this chapter we describe the most relevant details of the implementation of our solution. First, in section 4.1, we will see the most important aspects of the adaptation of the Client application. Then, in section 4.2, we will overview the Server node implementation and describe some of its most interesting details.

4.1 Client

In this work, we adapted a Latex editor called *Texmaker*. Since the application had no cooperation capabilities, the adaptation had two phases. The first served to adapt the application to a total consistency model which met the expected requirements. Only afterwards, this version was improved to incorporate VFC properties.

In this section we explain how the adaptation to the total consistency model and later to the VFC model was achieved.

4.1.1 Texmaker

The Texmaker is an open-source Latex editor application written in *C++* that can run in *Unix*, *Mac OS X*, and *Windows* systems. For this work we have adapted the Linux version. The editor is basically a text editor build using the *Qt*[1] GUI framework. Then, the edition capabilities were enhanced with modules for Latex structure recognition, text highlighting, text completion, and several other functionalities whose intent is to support the writing of a Latex document.

This application was chosen since it was a widely used open-source Latex editor which we could enhance to support concurrent editions with the desired freedom and, afterwards to support VFC enabled cooperation.

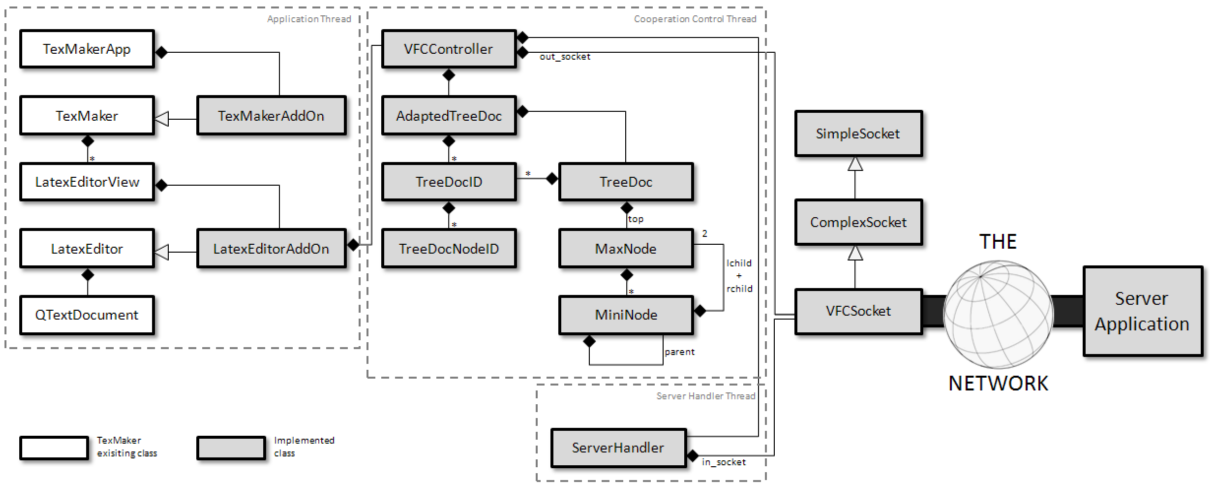


Figure 4.1: Client Class Diagram

Adaptation Results

We adapted the Texmaker editor to support cooperation using the VFC model (Figure 4.1). From this adaptation resulted two prototypes which provide total consistency and VFC capabilities. The intention of the creation of these two separated versions was to support the testing phase in such way that we had a comparison purpose prototype.

We can see from the class diagram that we totally separated the cooperation aspects (*Cooperation Control Thread*) form the user interface (*Application Thread*). This way, even in moments of greater cooperation stress, we are able safeguard the application usability.

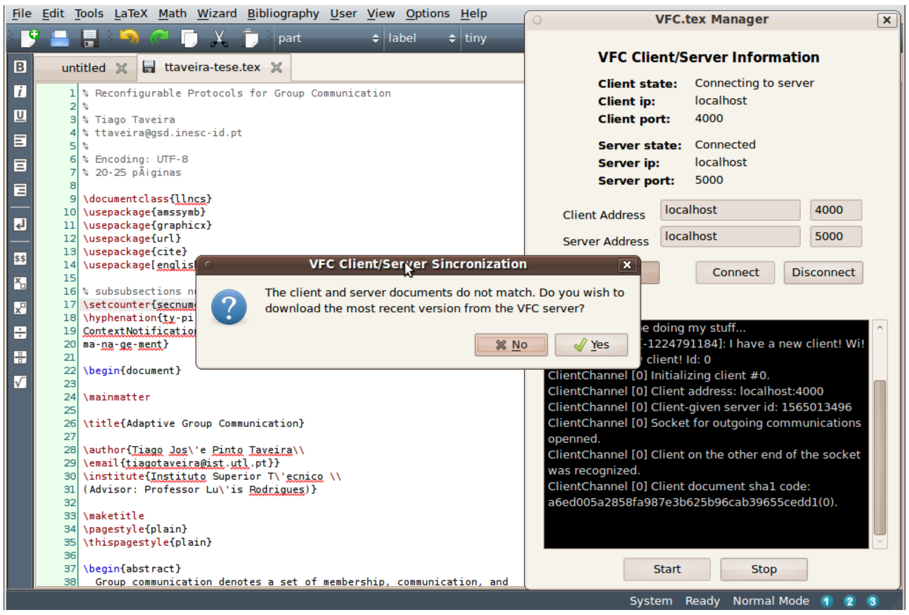
Making the connection with the defined architecture (section 3.2.2), we can easily perceive were the layers functionalities were implemented. The *TexMakerAddOn* is the class that makes a direct interaction with the user, thus, it is part of the *Application Enhancement Layer*. The functionality of this layer is also present in the *LatexEditorAddOn*. Then, the *VFC Location Protocol Layer* is implemented in the *VFCController* class. Also in the VFCController class, we can find all the Cooperation Protocol Layer implementation, apart of course from the *TreeDoc*, which is obviously implemented in the *AdaptedTreeDoc* class and in all the ones below.

User Interface

Using the Qt framework, we designed a window to control the launching of the server application and/or the establishment of connections to other servers. Also, a series of information dialogs to help the user solve any problems with the use of the VFC functionalities and of course, to help debugging.

In Figure 4.2 we can see an example of the application main window and *VFC.tex Manager* window while the user is trying to establish a connection with the server. A user familiar with the Texmaker application will see that, apart from the opened window (which can be closed at any moment), there are no visible changes to the application interface. In fact, the only change is a button on the fast access bar which either opens or closes the VFC.tex Manager window.

[1]http://qt.nokia.com/products/



```
% Reconfigurable Protocols for Group Communication
%
% Tiago Taveira
% ttaveira@gsd.inesc-id.pt
%
% Encoding: UTF-8
% 20-25 pÃiginas

\documentclass{llncs}
\usepackage{amssymb}
\usepackage{graphicx}
\usepackage{url}
\usepackage{cite}
\usepackage[englis

% subsubsections n
\setcounter{secnum
\hyphenation{ty-pi
ContextNotificatio
ma-na-ge-ment}

\begin{document}

\mainmatter

\title{Adaptive Group Communication}

\author{Tiago Jos\'e Pinto Taveira\\
\email{tiagotaveira@ist.utl.pt}}
\institute{Instituto Superior T\'ecnico \\
(Advisor: Professor Lu\'is Rodrigues)}

\maketitle
\pagestyle{plain}
\thispagestyle{plain}

\begin{abstract}
  Group communication denotes a set of membership, communication, and
```

VFC Client/Server Sincronization

The client and server documents do not match. Do you wish to download the most recent version from the VFC server?

No | Yes

VFC Client/Server Information

Client state:	Connecting to server
Client ip:	localhost
Client port:	4000
Server state:	Connected
Server ip:	localhost
Server port:	5000

Client Address	localhost	4000
Server Address	localhost	5000

```
e doing my stuff...
-1224791184]: I have a new client! Wi!
 client! Id: 0
ClientChannel [0] Initializing client #0.
ClientChannel [0] Client address: localhost:4000
ClientChannel [0] Client-given server id: 1565013496
ClientChannel [0] Socket for outgoing communications
openned.
ClientChannel [0] Client on the other end of the socket
was recognized.
ClientChannel [0] Client document sha1 code:
a6ed005a2858fa987e3b625b96cab39655cedd1(0).
```

Figure 4.2: Attempting to establish connection with the server

$$Add(pos, size) \longrightarrow Add(pos, content) \longrightarrow Add(id, content)$$
$$Remove(pos, size) \longrightarrow Remove(id, size)$$
$$Replace(pos, size, size) \longrightarrow Remove(pos, size) + Add(pos, size)$$

Figure 4.3: Adaptation of the captured operations into VFC operations

4.1.2 Interception of Updates

The Qt framework, already provides the means to intercept changes in the document (after they happen), in the form of add, remove and replace operations. However, it is frequent to have what we call fake updates. Fake updates are changes in the document that do not correspond exactly to what the user has done.

For example, when there was an insertion of characters with accent marks, the framework signaled a replace of the whole line for a line with the changes. Also, due to the text highlighting, many updates were signaled for replacement of the whole contents for equal contents. These fake updates, if not treated, would compromise the application overall performance since the number and size of operations would increase. Also, although this would not affect consistency, due to the tree ordering properties, it would compromise the user intentions. For example, a user editing in the middle of a line, would see his editions appear in the end of that line since the previous contents were deleted and placed again in the beginning of that same line.

Once the updates were filtered, some adaptations must be done to these operations so they can fit the TreeDoc model. In Figure 4.3, we can see how these captured operations were adapted.

4.1.3 Insertion of Received Updates

When updates arrive at the Client application, they need to be inserted in both the plain text and the TreeDoc representations of the document. Since they are controlled by two different threads, there is the need to have a kind of commitment protocol. In Figure 4.4 we can see in general terms how that

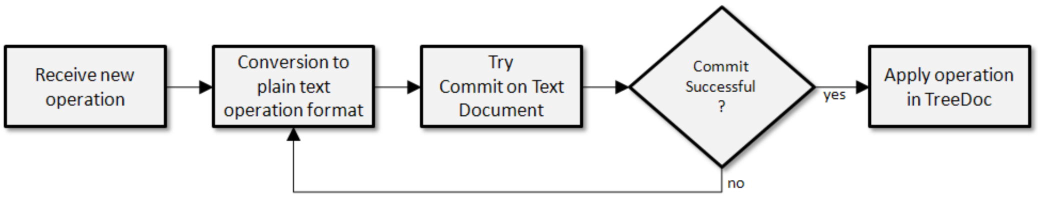

Receive new operation
Conversion to plain text operation format
Try Commit on Text Document
Commit Successful ?
yes
no
Apply operation in TreeDoc

Figure 4.4: Method for insertion of received updates

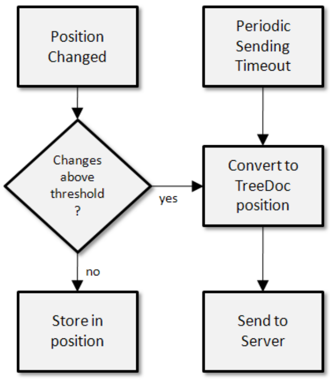


Figure 4.5: Interception of the user position

commitment is made.

Notice that the insertion in the TreeDoc is only done after the plain text representation is up-to-date. This is required since the thread which receives the new updates only controls the TreeDoc representation of the document.

4.1.4 Interception of the User Position

As explained in Chapter 3, there are two ways of informing the server about the user position: implicit and explicit message sending.

The implicit messages are already supported by a total consistent model. As for the explicit messages, in this particular case, they were not. So, we had to monitor the user position and intercept any changes. Fortunately, the Qt framework provided signals to the events of position changes. This way, we just need to keep an updated version of the user position known by the server (in form of a TreeDoc path/ID) which is periodically sent to it (in this case, every five seconds). To cope with sudden changes in position that could cause region changes, every time the position changes above a predetermined threshold (in this case, a 1000 characters threshold), the position is also sent to the server (see Figure 4.5).

4.2 Server

The server application, following the client adaptation was also written in C++. To promote its portability, we choose not to depend on libraries like *Boost*[2] or *Remote Call Framework* (RCF[3]) or any other similar purpose library. However, for the sake of future code understanding and easier debugging, we

[2]http://www.boost.org/
[3]http://code.google.com/p/rcf-cpp/

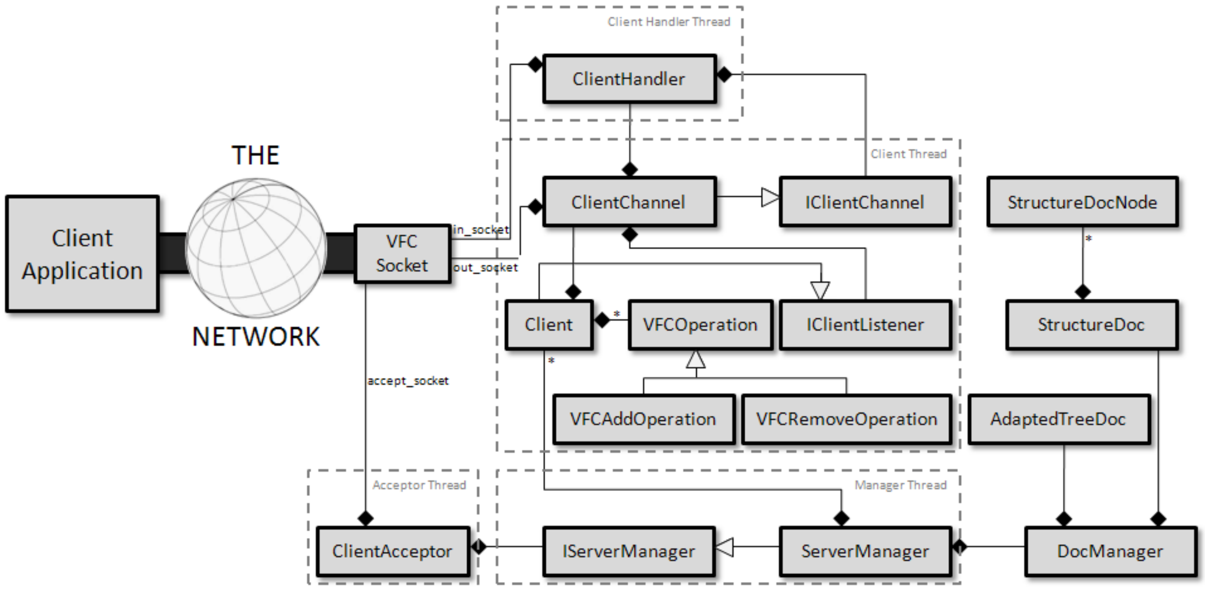


Figure 4.6: Server Class Diagram

implemented from scratch three class types: *Simple Sockets*, *Simple Threads* and *Simple Mutex*. For implementing these special class types we used, in the first case, *linux sockets* and in the remaining *pthreads*.

In terms of created classes, the server implementation, as we can see in Figure 4.6, follows the architecture presented in section 3.2.2. As we can see, there is a direct match between the architectural components and their class implementations.

Again, to have achieve better performance results we have several threads running at the same time. For example, with the separation of the *Manager Thread* and each *Client Thread*, we were able to prevent that temporary peaks in the Server Manager work, affected the connection with the Client nodes.

One final class which, due to not being directly related with the model, was not mentioned in the class diagram of Figure 4.1, is the *Statistics* class. This class is responsible for receiving and processing all the relevant information and, depending on the objectives of the tests, to generate the required evaluation results.

Chapter 5

Evaluation

The adaptation of the VFC model was evaluated in a qualitative (section 5.1), a quantitative (section 5.2) and a comparative (section 5.3) perspectives. In the first we argue about the success of the adaptation to the context of the document edition. The second consists in a study of the used network resources. The third and last is an argumentative comparison between the use of the VFC model and a Total Consistency (TC) model.

5.1 Qualitative Evaluation

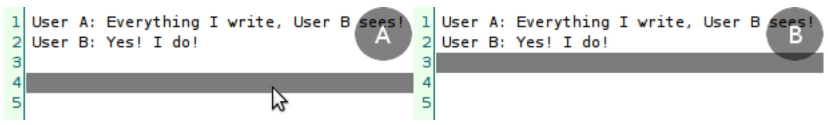
In this work we improved a total-consistent cooperative version of a Latex editor (*Texmaker*) to provide it with the benefits of the VFC model. The main objective of this qualitative study is to assess if the VFC model had a positive impact in the user interaction with the cooperative application.

To evaluate the quality impact we will see if the application usability was not compromised, i.e. if cooperation was not hindered by the introduction of the VFC model. Then we will see if the adapted VFC model is in fact capable of selecting the updates most related with the user editions.

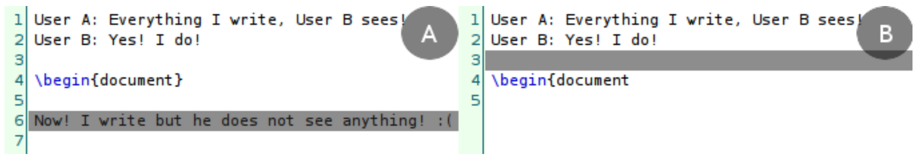
5.1.1 Evaluation of the Application Usability

A document editor using a VFC model has obvious differences when compared to one using a TC model. For instance when a user changes to a previously distant region, he will probably experience a sudden arrival of a series of updates. Also, he is supposed to notice that immediately around his cursor position there is an apparent larger activity rate than in further away regions.

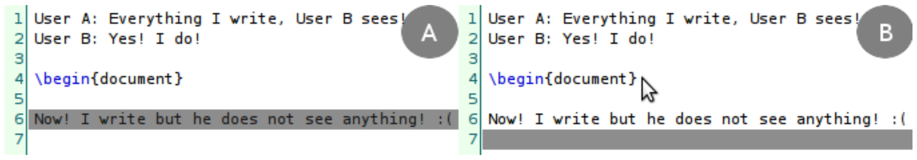
In practice, when using the adapted application, we can experience these exact situations. This gives the user a confidence that his region of edition is as much up-to-date as possible. Also, when testing the application with human users they were able to experience the enforcement of VFC limits and validate their advantages. Finally, there was no noticeable losses to the application performance when compared to the total consistent version. In fact, the types of delays found in the total consistent version were smoothed in the VFC version.


```
User A: Everything I write, User B sees!
User B: Yes! I do!
```
```
User A: Everything I write, User B sees!
User B: Yes! I do!
```

(a) Total Consistency with no Structure


```
User A: Everything I write, User B sees!
User B: Yes! I do!

\begin{document}

Now! I write but he does not see anything! :(
```
```
User A: Everything I write, User B sees!
User B: Yes! I do!

\begin{document
```

(b) Detection of Structure changes


```
User A: Everything I write, User B sees!
User B: Yes! I do!

\begin{document}

Now! I write but he does not see anything! :(
```
```
User A: Everything I write, User B sees!
User B: Yes! I do!

\begin{document}

Now! I write but he does not see anything! :(
```

(c) Detection of Region changes

Figure 5.1: Cooperation Scenario 1: Edition of an Empty Document

5.1.2 Evaluation of the VFC model Adaptation

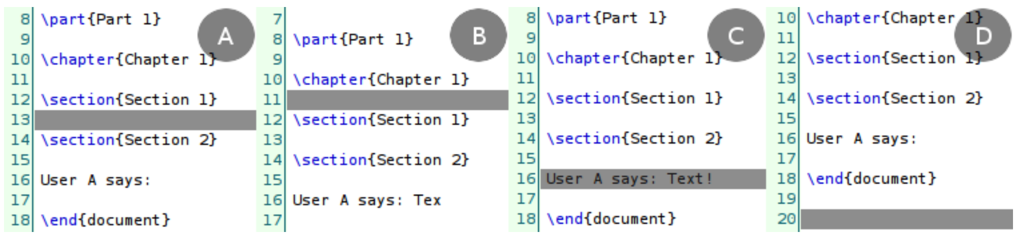
The adaptation to the VFC model, which was the great objective of this work, was successful. In fact, we were able to provide an enhanced cooperation environment where, using the already described location-aware techniques, there is a selective scheduling of updates according to their probable (or expected) relation (and relevance) with the user point of edition.

The following scenarios will be used to show that the scheduling of updates is in accordance to our expectations.

Cooperation Scenario 1: *Empty Document*

The first scenario shows what happens when two users start writing a document from scratch. Initially, since there is no structure, cooperation happens in a total consistent manner (Figure 5.1(a)). Then, when the first structural element appears, in this case, a "\begin{*document*}", users A and B will have new locations in the document structure. Since users A and B are now respectively on the regions *Preamble* and *Document*, the distance between their regions is maximal. Thus, their updates will only eventually be propagated to each other (Figure 5.1(b)). Now, if user B changes his position to the Document region, he will receive the new updates immediately (Figure 5.1(c)).

What we have shown with this example was that, when we have no structure, a total consistency model is applied. We have also illustrated the automatic recognition of structural changes and the effects of those changes in the delaying of updates propagation. Finally, we have demonstrated the detection of a location change that results in a shortening of distances and the consequences of that immediately reflected in the update management.



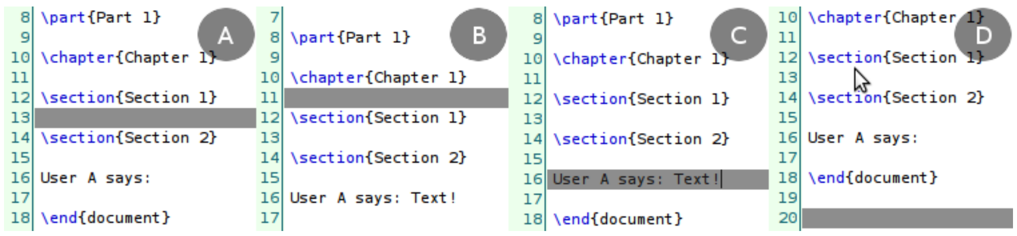
```
\part{Part 1}

\chapter{Chapter 1}

\section{Section 1}

\section{Section 2}

User A says:

\end{document}
```

```

\part{Part 1}

\chapter{Chapter 1}

\section{Section 1}

\section{Section 2}

User A says: Tex

```

```
\part{Part 1}

\chapter{Chapter 1}

\section{Section 1}

\section{Section 2}

User A says: Text!

\end{document}
```

```
\chapter{Chapter 1}

\section{Section 1}

\section{Section 2}

User A says:

\end{document}

```

(a) Immediately after insertion of "*Text!*"



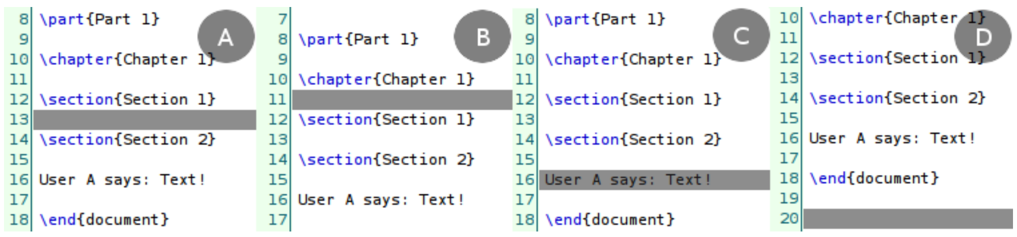
```
\part{Part 1}

\chapter{Chapter 1}

\section{Section 1}

\section{Section 2}

User A says:

\end{document}
```

```

\part{Part 1}

\chapter{Chapter 1}

\section{Section 1}

\section{Section 2}

User A says: Text!

```

```
\part{Part 1}

\chapter{Chapter 1}

\section{Section 1}

\section{Section 2}

User A says: Text!

\end{document}
```

```
\chapter{Chapter 1}

\section{Section 1}

\section{Section 2}

User A says:

\end{document}

```

(b) Region 1 Timeout



```
\part{Part 1}

\chapter{Chapter 1}

\section{Section 1}

\section{Section 2}

User A says: Text!

\end{document}
```

```

\part{Part 1}

\chapter{Chapter 1}

\section{Section 1}

\section{Section 2}

User A says: Text!

```

```
\part{Part 1}

\chapter{Chapter 1}

\section{Section 1}

\section{Section 2}

User A says: Text!

\end{document}
```

```
\chapter{Chapter 1}

\section{Section 1}

\section{Section 2}

User A says: Text!

\end{document}

```

(c) Region 4 Timeout

Figure 5.2: Cooperation Scenario 2: *Consistency Zone Timeouts*

Cooperation Scenario 2: *Time Limits Enforcement*

The second scenario has the objective of demonstrating the enforcement of VFC time limits.

In this scenario, we start with four users cooperatively editing a document with only a Part, a Chapter and two Sections. In this example, all users are synchronized. However, they are located in different positions in a way that user C is at a distance 1 from user B, 3 from user A and 4 from user D. Then, user C inserts "*Text!*" and waits (Figure 5.2(a)). What happens is that, after approximately 8 seconds (Figure 5.2(b)), user B (the closest) becomes consistent with user C. Later, after approximately 89 seconds, user A receives the updates. And finally, only after approximately 268 seconds (almost 5 minutes), the updates are delivered to user D (Figure 5.2(c)).

With this scenario, we have shown how distances between users points of edition affect the time elapsed between the generation of updates and their arrival to each user. Notice that in any of these cases the time limits were exceeded.

```
#* E V E N T *# Client [5]: New Operation in Region 6!
Client [5]: Merge ADD/REM results: 15/15
Client [5]: Distance between client @ R4 and R6 is 1.
Client [5]: Checking Sequence: NumOps >= SEQ[1]: 15 >= 15.
Client [5]: Delivering all operations for region 6.
Client [5]: Merge ADD/ADD & REM/REM results: 1/15
```

Figure 5.3: Cooperation Scenario 3: Exert of the Server Log showing delivery decision: $15 \geqslant 15$

```
#* E V E N T *# Client [7]: New Operation in Region 6!
Client [7]: Merge ADD/REM results: 100/100
Client [7]: Distance between client @ R3 and R6 is 2.
Client [7]: Checking Sequence: NumOps >= SEQ[2]: 100 >= 100.
Client [7]: Delivering all operations for region 6.
Client [7]: Merge ADD/ADD & REM/REM results: 50/100
```

Figure 5.4: Cooperation Scenario 3: Exert of the Server Log showing delivery decision: $100 \geqslant 100$

Cooperation Scenario 3: *Sequence Limits Enforcement*

This scenario has the objective of demonstrating the enforcement of VFC sequence limits. To achieve this objective, we defined two resembling subscenarios.

Cooperation Subscenario 3.1 We start with two users located in two different regions at distance 1 from each other. Then, user A inserts 14 characters, one at a time, without any of these being seen by user B. Only when the 15^{th} character is inserted and the sequence limits for that region are reached, user B receives the updates (containing all the inserted characters) and gets in synch with user A (Figure 5.3).

Cooperation Subscenario 3.2 We start with two users located in two different regions at distance 2 from each other. Then, user A removes 99 characters, one at a time, without any of these changes being seen by user B. Only when the 100^{th} character is removed and the sequence limits for that region are reached, user B receives the updates and gets in synch with user A (Figure 5.4).

In these examples, we have shown how the arrival of new operations affects the VFC sequence limits in two different consistency zones and how, when reached, they will cause the delivery of all updates in a specific region.

Cooperation Scenario 4: *Value Limits Enforcement*

In this scenario we intend to evidence the enforcement of VFC value limits. To do it, we propose a scenario where user A repeatedly makes insertions of 10 characters all at once, until the updates are propagated to user B.

The scenario was experimented in three different situations, where users A and B were in regions at a distance of respectively 1, 2 and 3. These differences would be visible in the number of insertions needed to surpass the value limits.

```
#* E V E N T *# Client [3]: New Operation in Region 6!
Client [3]: Merge ADD/REM results: 5/5
Client [3]: Distance between client @ R3 and R6 is 2.
Client [3]: Checking Sequence: NumOps >= SEQ[2]: 5 >= 100.
Client [3]: Checking Value: ChangedChars(x100) >= VAL[2]*RegionSize: 5000 >= 4950.
Client [3]: Delivering all operations for region 6.
Client [3]: Merge ADD/ADD & REM/REM results: 5/5
```

Figure 5.5: Cooperation Scenario 4: Exert of the Server Log showing delivery decision: $50 \geqslant 49{,}5$

```
#* E V E N T *# Client [15]: New Operation in Region 5!
Client [15]: Merge ADD/REM results: 71/71
Client [15]: Distance between client @ R4 and R5 is 3.
Client [15]: Checking Sequence: NumOps >= SEQ[3]: 71 >= 750.
Client [15]: Checking Value: ChangedChars(x100) >= VAL[3]*RegionSize: 17900 >= 17700.
Client [15]: Delivering all operations for region 5.
Client [15]: Merge ADD/ADD & REM/REM results: 13/71
```

Figure 5.6: Cooperation Scenario 4: Exert of the Server Log showing delivery decision: $179 \geqslant 177$

In this first case, right after the first insertion, the limits were surpassed and the contents of users A and B became consistent. In the second case (Figure 5.5), user A had to make 5 insertions before the updates were propagated. At last, in the third case (Figure 5.6) only after 18 insertions, the value limits were exceeded.

In conclusion, this scenario demonstrated how the arrival of new operations might result in the surpassing of the value limits and how those limits vary depending on the distance between the locations of users and editions. Also, we demonstrated the difference between the *value* and *sequence* limits. In this scenario, although we inserted so much characters, the sequence limits were never reached. This happens because, when the user pastes a series of characters, it counts as only one operation. Thus, although we inserted a great number of characters, the number of operations was not so high.

5.2 Quantitative Evaluation

In this section we show the results of a series of conducted tests. These tests had the objective of determining the savings in the use of network resources. None of the performed tests were repeated, so there was no selection of the most convenient results.

In these tests we used two machines, one running only the server and the second running a predetermined number of bots, depending on the objective, with the intention of simulating a certain edition scenario. The presented results are obviously only representative of the edition patterns of the writer bots, which are not exactly those of a normal human editor, but the ones that can better test our system. We describe next the writer bots to provide some perspective on the parameters that rule their behavior and types of actions that can be recreated.

Finally, the tests were executed in an application powered with the VFC model and the results for the TC model were inferred by a special-purpose module during each test conduction.

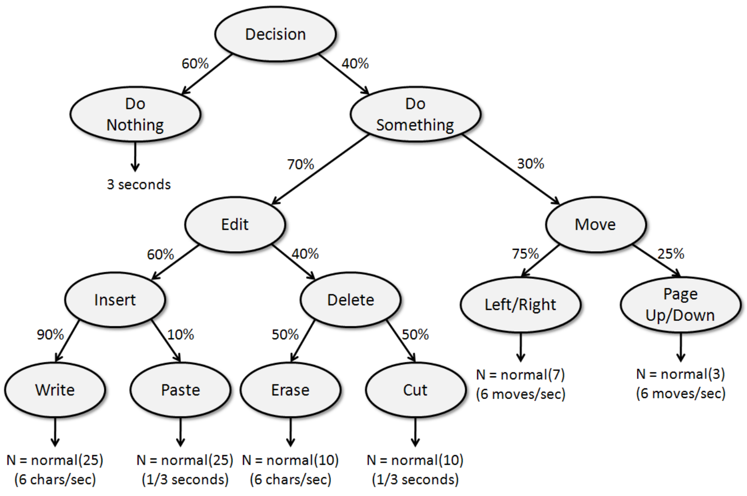
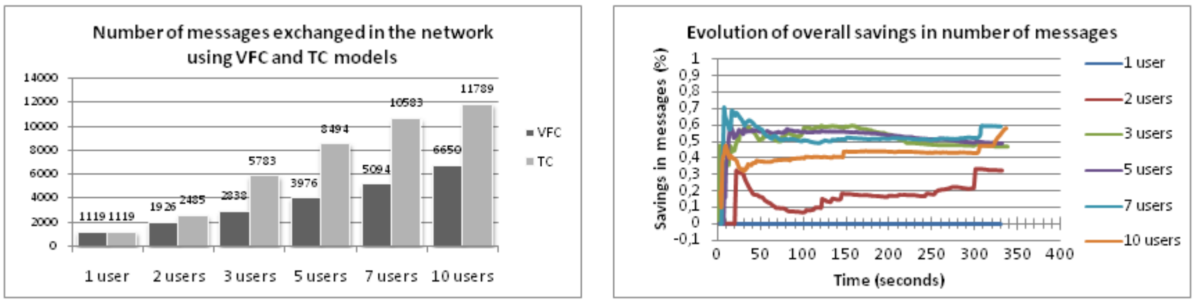

Decision
60%
40%
Do Nothing
3 seconds
Do Something
70%
30%
Edit
Move
60%
40%
75%
25%
Insert
Delete
Left/Right
Page Up/Down
N = normal(7) (6 moves/sec)
N = normal(3) (6 moves/sec)
90%
10%
50%
50%
Write
Paste
Erase
Cut
N = normal(25) (6 chars/sec)
N = normal(25) (1/3 seconds)
N = normal(10) (6 chars/sec)
N = normal(10) (1/3 seconds)

Figure 5.7: Decision tree to simulate Writer Bot behavior



(a) Comparing VFC and TC models

(b) Evolution of overall savings

Figure 5.8: Number of messages exchanged in the network

5.2.1 Writer Bot

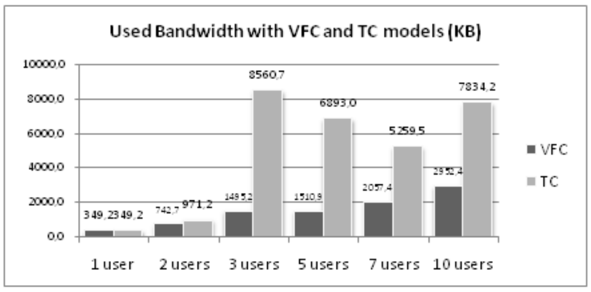
A writer bot is a test-oriented software that generates updates. They worked in cycle taking decisions based only on a *decision tree* travelled by the means of a probability test. Then, the behavior of the writer bot could be adjusted by balancing the probabilities that influence the decision in each tree branch.

With the used parameters (Figure 5.7) we were able to simulate fast typing users in an intensive edition moment. Other than a slightly exaggerated rate of edition, since the bots were created to test the prototype, not to act exactly like humans, the changes in locations are more frequent, and the pauses in actual edition are shorter than usual.

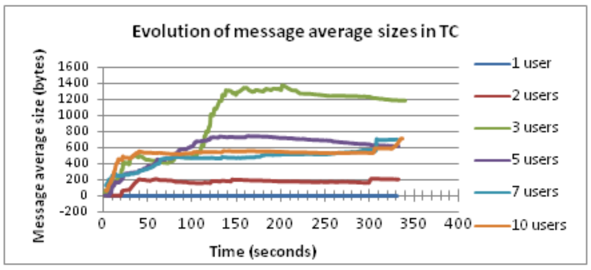
5.2.2 Number of Messages

In this work, as far as we can conclude from the experimental results, we were able to have overall reductions in the number of exchanged messages of about 50% when with an already interesting number of users, as can be seen in Figure 5.8. Of course, when we only have 1 user, we do not have any gains.

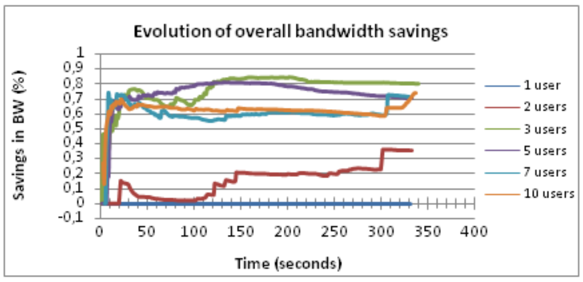
Another interesting conclusion was that with these savings, the impacts of increasing the number of users from 3 to 10 users were very little in terms of number of exchanged messages.



(a) Comparing VFC and TC models



(b) Evolution of message average sizes in TC



(c) Evolution of overall savings

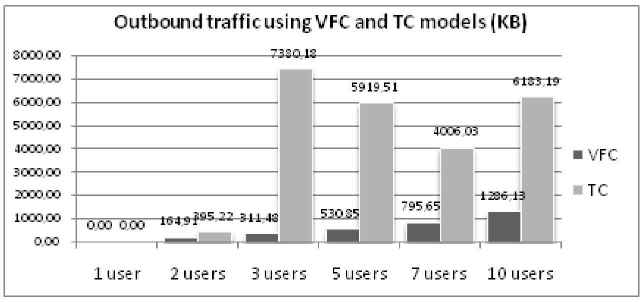
Figure 5.9: Bandwidth savings in the network

In the results we see that from the moment that we have more than two users, that we start having much greater savings in the number of exchanged messages. Also, when editing a document with only two users, the evolution of the overall savings is less predictable than with another number of users. This can be justified by the great impact that changes in a user position have in all the distances in this type of scenario.

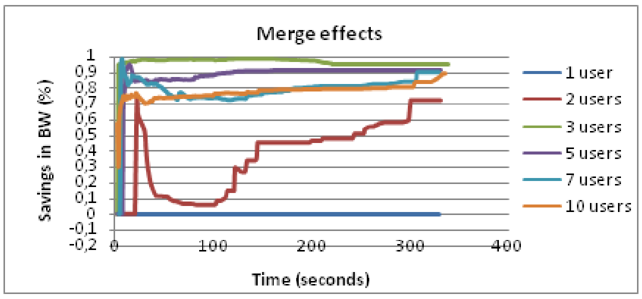
Finally, a comment about the sudden increase in the overall savings in exchanged messages short after the initial five minutes. This increase, coherent with the expected trigger time of the timeout signal for Consistency Zone 4, shows the beneficial effect of the delay and merge of operations.

5.2.3 Used Bandwidth

The results obtained in these series of tests in terms of used bandwidth are deceiving, and apparently contradicting with the results for the number of exchanged messages. These odd results, seen for example



(a) Comparing VFC and TC models



(b) Evolution of overall merge savings

Figure 5.10: Outbound traffic merge savings

in the test with 3 users 5.9(a), can be explained by moments in which there was an inflation of the path sizes due to the tail problem (section 3.2.3) and, consequently, in the size of message (5.9(b)).

Even though the *tail problem* had so much impact in the results for the Total Consistent model, we can see in Figure 5.9(a) that, with the VFC model, we were able to damp those effects and control the used bandwidth.

As seen by the savings in the overall used bandwidth, for the tests involving more than two users, we were able to have final savings of around 70%. And even though this only happened after the last consistency zone timeout, we can still say that since the first seconds of the test, the overall savings were never below 50%.

5.2.4 Operation Merge Results

The VFC model generates benefits when updates are delayed and merged with each other before being sent. Thus, we now show how efficient were those merge operations, which affected only the server outbound traffic.

Looking at the results in Figure 5.10 we see that the merge operations were able to compress the operations to 90% of the original size, which shows the real potential of the use of the VFC model.

Additionally, these results show the efficiency of the solution used to solve the tail problem and how that solution can be used to generate great savings in the memory required to store a document in form of a TreeDoc.

Average VFC measurements					
Zone	**#Regions**	**#Ops**	**Time**	**Sequence**	**Value**
0	398	684	0,34s	1,72 un.	0,48%
1	42	989	1,93s	23,55 un.	7,67%
2	4	369	9,97s	92,25 un.	20,68%
3	3	98	20,14s	32,67 un.	19,43%
4	7	465	259,85s	66,43 un.	26,44%
ALL	454	2605	49,38s	5,74 un.	1,85%

Table 5.1: Average VFC measurements

5.2.5 VFC Limits

In Table 5.1, we can see the measurements of the average *time* (imposed delay for each operation), *sequence* (number of operations) and *value* (ratio between the changes and the region size) of all the updates in a session of 5 users during 5 minutes.

The first conclusion is that the updates in the Consistency Zone 0, i.e. around the user location of edition, take an average of 0,34s to be transmitted, which is sufficiently fast not to hinder the usability of the application. Also, about the average values for the time measurements, we can see that they are much below the maximum limits. This means that, even if there is a temporary situation with a peak in the number of incoming updates, it can be rapidly compensated. Another interesting fact is that only the average delay for the Consistency Zone 4 comes close to its maximum limit. This can be explained by the fact that it is not possible to have structural changes causing sudden reach of VFC limits in this zone.

Notice that the number of updated regions in the outermost consistency zones is smaller then predicted. For this reason, the variance of the average value measure is not significant. These results can be easily explained: the bigger the divergence limits, the bigger the time elapsed before the operation is sent. During that time, the user can change his position, which changes the consistency zones and may result in the need to immediately propagate the updates. Even if that does not happen, the longer the update is retained, the greater is the probability of having updates cancelling each other, which will delay even more their sending. Also, their average sequence and value measurements are way below the limits (except in one case) which means that eventually the maximum time limit was reached and the updates were sent.

5.3 Comparative Evaluation

In the previous sections we shown the results of several conducted tests. Those tests proved that the VFC model has great benefits when compared with the TC model. The bigger the delay imposed to the updates, the more savings we were able to generate.

In conclusion, unless the the selective delaying of updates will hinder the user experience, which is not an expected scenario, in comparison to the TC model, VFC has a superior management of network resources and, for that reason, will be able to scale larger numbers of users.

Chapter 6

Conclusion

In this paper we have presented a *continuous consistency model* which results from the adaptation of the *VFC* model to the scenario of cooperative work, namely of the cooperative edition of documents. The VFC model, besides providing a continuum between strong and weak consistency, is combined with the notion of *locality awareness*, which provides an enhanced cooperation experience where the user is selectively updated in accordance with the contents under edition.

To begin this work, we overviewed the state of the art of two important fields: *consistency maintenance in distributed systems* and *cooperative work tools.*

Then, and most importantly, we defined the architecture for a system powered with the adaptation of the VFC model. Here, we defined the required adaptations that, in our opinion, are required to best adjust the VFC model to the new scenario of cooperative work, specifically document-based cooperative work. In this adaptation, we redefined the concepts of the *user's location* (*pivot*), *distance* between pivots and replicated objects, and *consistency zones.* In this case, we supported the definitions on the notion of *document structure.* Finally, the adaptation also involved porting the update representation from a state-transfer to an *operation-transfer* solution. In fact, to provide the freedom of edition desired in such context, the cooperation was based on a *CRDT*, namely on a *TreeDoc* representation.

The defined architecture was implemented on top of a totally consistent system, also developed in this work. After the implementation, using a number of criteria, we analyzed the success of this work. Other than showing the correctness of the adaptation, the results showed the great potential of the VFC model. In comparison to the use of the TC model, we were able to achieve large reductions in the use of network resources. As such, a VFC powered system provides the enhanced cooperation scenario that we cannot find in the most popular document editors.

6.1 Future Work

In this work, because of the complexity of the implementation of both the TC and the VFC systems, we were not able to experiment and optimize our solutions as much as we wanted. We now address some of the future directions that this work may take.

Use of Efficient Update Representations

As seen in the related work, in section 2.1.6 there are some interesting alternatives to the use of state-transfer and operation-transfer in which we mix these two solutions. If we think in the particular case of this work, it could be interesting to try to use state-transfer solutions in the most distant regions of the document. This way, we could possibly achieve great bandwidth savings. However, it is not that clear that this solution would not hinder the usability of the application, since, when switching from a state-transfer to an operation-transfer in a certain region, there could be important performance delays because of the need to retrieve the most recent TreeDoc representation of that zone from the server.

Fully Distributed Version

Right now we are centralizing all the VFC reasoning in a single node, the Server. Although it has the advantage of taking the most correct decisions, in the current solution, every communication has to go trough this node. Thus, when the number of users or operations increases it might still become an important bottleneck in the system. In short, it would be interesting to try to create a fully distributed VFC model, where every node had a similar role, and the VFC reasoning was itself carry out in a distributed manner.

Intra-Document References

Unfortunately, we were not able to explore the benefits of the intra-document references (section 2.2.3). Using these references we could improve the user experience by propagating the updates more quickly in regions which were distant in terms of the document structure but highly related in terms of their semantics. This way, we would provide a more accurate inference about the dependences between the contents of the various regions.

Project Cooperation

In a Latex, the final document is frequently generated using the contents of several files. An interesting step in this work would be to provide cooperation not in a per-document basis but for whole projects, i.e. for a series of files that represent a single document. Even though the physical separation of contents in several files suggests that contents in different files are unrelated, if we use distance shortening commands, like references, the previous assumption fails.

Other Cooperation Environments

Finally, in this work we wanted to address the adaptation of the VFC model to other environments of the cooperative work, namely the cooperative edition of spreadsheets and presentations. In both cases, the adaptation seems to be smoother than the adaptation made in this work since the use of spacial definitions for distances between objects seems to be adequate to the characteristics of the environment.

Bibliography

[1] L. J. Bannon and K. Schmidt. Cscw: four characters in search of a context. In *Studies in computer supported cooperative work: theory, practice and design*, pages 3–16, Amsterdam, The Netherlands, The Netherlands, 1991. North-Holland Publishing Co.

[2] J. Barreto and P. Ferreira. A highly available replicated file system for resource-constrained windows ce .net devices. In *In 3rd International Conference on .NET Technologies*, 2005.

[3] J. P. Barreto, J. Garcia, L. Veiga, and P. Ferreira. Data-aware connectivity in mobile replicated systems. In *MobiDE*, pages 9–16, 2009.

[4] M. Bento and N. Preguiça. Operational transformation based reconciliation in the few file system. In *Proceedings of the Eight International Workshop on Collaborative Editing Systems - part of the ACM CSCW 2006*, 2006.

[5] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency control and recovery in database systems.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1987.

[6] M. Buffa, G. Crova, F. G, C. Lecompte, and J. Passeron. Sweetwiki: Semantic web enabled technologies in wiki. In *Proceedings of the 2006 international symposium on Wikis WikiSym '06; ACM*, pages 69–78. Press, 2006.

[7] M. Cart and J. Ferrie. Asynchronous reconciliation based on operational transformation for p2p collaborative environments. In *COLCOM '07: Proceedings of the 2007 International Conference on Collaborative Computing: Networking, Applications and Worksharing*, pages 127–138, Washington, DC, USA, 2007. IEEE Computer Society.

[8] U. Çetintemel, P. J. Keleher, B. Bhattacharjee, and M. J. Franklin. Deno: A decentralized, peer-to-peer object-replication system for weakly connected environments. *IEEE Trans. Comput.*, 52(7):943–959, 2003.

[9] D. J. Dietterich. Dec data distributor: for data replication and data warehousing. In *SIGMOD '94: Proceedings of the 1994 ACM SIGMOD international conference on Management of data*, page 468, New York, NY, USA, 1994. ACM.

[10] D. Eastlake, 3rd and P. Jones. Us secure hash algorithm 1 (sha1). United States, 2001. RFC Editor.

[11] C. A. Ellis and S. J. Gibbs. Concurrency control in groupware systems. In *SIGMOD '89: Proceedings of the 1989 ACM SIGMOD international conference on Management of data*, pages 399–407, New York, NY, USA, 1989. ACM.

[12] P. T. Eugster, R. Guerraoui, S. B. Handurukande, P. Kouznetsov, and A.-M. Kermarrec. Lightweight probabilistic broadcast. *ACM Trans. Comput. Syst.*, 21(4):341–374, 2003.

[13] I. Greif, editor. *Computer-supported cooperative work: a book of readings.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[14] J. Grudin. Why cscw applications fail: problems in the design and evaluationof organizational interfaces. In *CSCW '88: Proceedings of the 1988 ACM conference on Computer-supported cooperative work*, pages 85–93, New York, NY, USA, 1988. ACM.

[15] J. Grudin. Groupware and social dynamics: eight challenges for developers. *Commun. ACM*, 37(1):92–105, 1994.

[16] C.-L. Ignat and M. C. Norrie. Customizable collaborative editor relying on treeopt algorithm. In *ECSCW'03: Proceedings of the eighth conference on European Conference on Computer Supported Cooperative Work*, pages 315–334, Norwell, MA, USA, 2003. Kluwer Academic Publishers.

[17] F. Kappe. A scalable architecture for maintaining referential integrity in distributed information systems. In *JUCS 1 (2)*, 1995.

[18] N. Krishnakumar and A. J. Bernstein. Bounded ignorance: a technique for increasing concurrency in a replicated system. *ACM Trans. Database Syst.*, 19(4):586–625, 1994.

[19] T. Kuhn. Acewiki: A natural and expressive semantic wiki. *CoRR*, abs/0807.4618, 2008.

[20] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.

[21] S. Lawrence, F. Coetzee, E. Glover, D. Pennock, G. Flake, F. Nielsen, B. Krovetz, A. Kruger, and L. Giles. Persistence of web references in scientific research. *IEEE COMPUTER*, 34(2):26–31, 2001.

[22] J. C. Morris. Distriwiki:: a distributed peer-to-peer wiki network. In *WikiSym '07: Proceedings of the 2007 international symposium on Wikis*, pages 69–74, New York, NY, USA, 2007. ACM.

[23] A. Muthitacharoen, B. Chen, and D. Mazières. A low-bandwidth network file system. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 174–187, New York, NY, USA, 2001. ACM.

[24] D. A. Nichols, P. Curtis, M. Dixon, and J. Lamping. High-latency, low-bandwidth windowing in the jupiter collaboration system. In *UIST '95: Proceedings of the 8th annual ACM symposium on User interface and software technology*, pages 111–120, New York, NY, USA, 1995. ACM.

[25] S. Noël and J.-M. Robert. Empirical study on collaborative writing: What do co-authors do, use, and like? *Comput. Supported Coop. Work*, 13(1):63–89, 2004.

[26] G. Oster, P. Molli, S. Dumitriu, and R. Mondejar. Uniwiki: A collaborative p2p system for distributed wiki applications. In *WETICE '09: Proceedings of the 2009 18th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises*, pages 87–92, Washington, DC, USA, 2009. IEEE Computer Society.

[27] G. Oster, P. Urso, P. Molli, and A. Imine. Data consistency for p2p collaborative editing. In *CSCW '06: Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, pages 259–268, New York, NY, USA, 2006. ACM.

[28] N. Preguiça, J. M. Marques, M. Shapiro, and M. Letia. A commutative replicated data type for cooperative editing. In *ICDCS '09: Proceedings of the 2009 29th IEEE International Conference on Distributed Computing Systems*, pages 395–403, Washington, DC, USA, 2009. IEEE Computer Society.

[29] C. Rahhal, H. Skaf-Molli, and P. Molli. Swooki: A peer-to-peer semantic wiki. In *SemWiki*, 2008.

[30] J. Rama and J. Bishop. A survey and comparison of cscw groupware applications. In *SAICSIT '06: Proceedings of the 2006 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries*, pages 198–205, , Republic of South Africa, 2006. South African Institute for Computer Scientists and Information Technologists.

[31] M. Raynal and M. Singhal. Logical time: Capturing causality in distributed systems. *Computer*, 29(2):49–56, 1996.

[32] T. Rodden. A survey of cscw systems. *Interacting with Computers*, 3:319–353, 1992.

[33] H.-G. Roh, J. Kim, and J. Lee. How to design optimistic operations for peer-to-peer replication. In *JCIS*, 2006.

[34] Y. Saito and M. Shapiro. Optimistic replication. *ACM Comput. Surv.*, 37(1):42–81, 2005.

[35] N. Santos, L. Veiga, and P. Ferreira. Vector-field consistency for ad-hoc gaming. In *Middleware '07: Proceedings of the ACM/IFIP/USENIX 2007 International Conference on Middleware*, pages 80–100, New York, NY, USA, 2007. Springer-Verlag New York, Inc.

[36] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere. Coda: A highly available file system for a distributed workstation environment. *IEEE Trans. Comput.*, 39(4):447–459, 1990.

[37] M. Shapiro and N. Preguia. Designing a commutative replicated data type. Technical report, Computer Science Dept: University of Copenhagen, 2007.

[38] C. Sun and C. Ellis. Operational transformation in real-time group editors: issues, algorithms, and achievements. In *CSCW '98: Proceedings of the 1998 ACM conference on Computer supported cooperative work*, pages 59–68, New York, NY, USA, 1998. ACM.

[39] C. Sun, Y. Yang, Y. Zhang, and D. Chen. A consistency model and supporting schemes for real-time cooperative editing systems. In *Proceedings of the 19th Australian Computer Science Conference (Melbourne, Jan.)*, pages 582–591, 1996.

[40] D. Sun, S. Xia, C. Sun, and D. Chen. Operational transformation for collaborative word processing. In *CSCW '04: Proceedings of the 2004 ACM conference on Computer supported cooperative work*, pages 437–446, New York, NY, USA, 2004. ACM.

[41] S. G. Tammaro, J. N. Mosier, N. C. Goodwin, and G. Spitz. Collaborative writing is hard to support: A field study of collaborative writing. *Comput. Supported Coop. Work*, 6(1):19–51, 1997.

[42] D. B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser. Managing update conflicts in bayou, a weakly connected replicated storage system. In *SOSP '95: Proceedings of the fifteenth ACM symposium on Operating systems principles*, pages 172–182, New York, NY, USA, 1995. ACM.

[43] L. Veiga and P. Ferreira. Repweb: replicated web with referential integrity. In *SAC '03: Proceedings of the 2003 ACM symposium on Applied computing*, pages 1206–1211, New York, NY, USA, 2003. ACM.

[44] L. Veiga and P. Ferreira. Semantic-chunks a middleware for ubiquitous cooperative work. In *ARM '05: Proceedings of the 4th workshop on Reflective and adaptive middleware systems*, New York, NY, USA, 2005. ACM.

[45] M. Völkel, M. Krötzsch, D. Vrandecic, H. Haller, and R. Studer. Semantic wikipedia. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 585–594, New York, NY, USA, 2006. ACM.

[46] S. Weiss, P. Urso, and P. Molli. Wooki: a p2p wiki-based collaborative writing tool. In *WISE'07: Proceedings of the 8th international conference on Web information systems engineering*, pages 503–512, Berlin, Heidelberg, 2007. Springer-Verlag.

[47] S. Weiss, P. Urso, and P. Molli. An undo framework for p2p collaborative editing. In *CollaborateCom*, pages 529–544, 2008.

[48] S. Weiss, P. Urso, and P. Molli. Logoot: A scalable optimistic replication algorithm for collaborative editing on p2p networks. In *ICDCS '09: Proceedings of the 2009 29th IEEE International Conference on Distributed Computing Systems*, pages 404–412, Washington, DC, USA, 2009. IEEE Computer Society.

[49] S. Xia, D. Sun, C. Sun, D. Chen, and H. Shen. Leveraging single-user applications for multi-user collaboration: the coword approach. In *CSCW '04: Proceedings of the 2004 ACM conference on Computer supported cooperative work*, pages 162–171, New York, NY, USA, 2004. ACM.

[50] H. Yu and A. Vahdat. Design and evaluation of a continuous consistency model for replicated services. In *OSDI'00: Proceedings of the 4th conference on Symposium on Operating System Design & Implementation*, pages 21–21, Berkeley, CA, USA, 2000. USENIX Association.

[51] H. Yu and A. Vahdat. The costs and limits of availability for replicated services. *ACM Trans. Comput. Syst.*, 24(1):70–113, 2006.