HAL
archives-ouvertes.fr

GroupDiv: Formalizing and Computing Group Divergence Awareness in Multi-Synchronous Distributed Collaborative Systems

Khaled Aslan-Almoubayed, Hala Skaf-Molli, Pascal Molli

▶ To cite this version:

Khaled Aslan-Almoubayed, Hala Skaf-Molli, Pascal Molli. GroupDiv: Formalizing and Computing Group Divergence Awareness in Multi-Synchronous Distributed Collaborative Systems. 2013. <hal-00842714>

HAL Id: hal-00842714
http://hal.univ-nantes.fr/hal-00842714

Submitted on 9 Jul 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

GroupDiv: Group Divergence Awareness for Distributed Multi-Synchronous Collaborative Systems

Khaled Aslan[a,*], Hala Skaf-Molli[a], Pascal Molli[a]

[a] *LINA, Nantes University, BP 92208, 44322 Nantes Cedex 03, France*
Phone: +33 2 51 12 58 17 Fax: +33 2 51 12 58 97

Abstract

Collaboration can be synchronous, asynchronous or multi-synchronous. In multi-synchronous collaboration, participants work in parallel on their own copies and synchronize periodically to build a consistent state. A multi-synchronous collaboration introduces divergence between copies of shared objects; collaboration is cycles of convergence/divergence.

In existing divergence awareness systems, divergence is viewed as conflicts introduced by concurrent modifications. However, divergence can exist without conflict, whenever an operation is produced by a participant and not consumed by other ones, divergence exists. In this paper, we propose a generic model to formalize multi-synchronous collaboration. Using this model, we formalize existing divergence awareness and we define an original divergence metric GroupDiv that quantifies divergence as the editing distance of the group to the next potential convergence point. This metric enables users to control cycles of convergence/divergence i.e. they cannot control the magnitude of divergence and the frequency of synchronization to achieve convergence. The evaluation shows that GroupDiv enables participants to manage the cycle of convergence/divergence.

Keywords: Awareness in Collaboration Systems, Collaboration Enabling Technologies, Platforms, Artifacts and Tools for Collaboration, Coordination and Cooperation Mechanisms, Simulation of Collaboration Systems, Web Infrastructure for Collaborative Applications

1. Introduction

Distributed collaborative systems allow people to work distributed in time, space and across organizations. Collaboration can be synchronous, asynchronous or multi-synchronous [1]. In multi-synchronous collaboration, participants work in parallel on their own copies and synchronize periodically to build a consistent state. Version control systems (CVS, SVN, git) and synchronizers (Dropbox, isync) are examples of multi-synchronous collaboration software. The multi-synchronous collaboration introduces divergence between copies of shared objects. If working in parallel can potentially reduce completion time, it induces blind modifications [2]. The overhead of solving conflicts introduced by concurrent modifications can overwhelm the expected gain [3, 4, 5]. Divergence awareness [6] makes participants aware of the quantity and the location of divergence in shared objects. It allows to answers the following questions: is there any divergence? With whom? Where? And how much? Divergence awareness is an implicit coordination mechanism [7, 8], it incites participants to coordinate their actions to reduce divergence. It can be provided by different systems, relying on different metrics with different ad-hoc visualizations like: State Treemap [9], Operational Transformation Divergence [6], Palantir [10], Edit Profile [11], Concurrent modifications [12], CollabVS [13], and Crystal [14].

*Principal corresponding author
Email addresses: khaled.aslan-almoubayed@univ-nantes.fr (Khaled Aslan), hala.skaf@univ-nantes.fr (Hala Skaf-Molli), pascal.molli@univ-nantes.fr (Pascal Molli)

Different approaches exist for computing divergence: estimating the size of conflicts [6], estimating the difference between users' copies and a reference copy [9], or estimating divergence according to multiple copies of reference [14]. Some systems only consider published operations [11], others consider unpublished operations [2]. In both cases, metrics can be projected according to different perspectives such as the structure of documents, the users, or across the time. This generates different kinds of visualization.

A first issue concerns divergence quantification. Even if existing divergence metrics are able to notify users about the presence of divergence and where it is located, they fail to clearly quantify divergence. If multi-synchronous collaboration systems are characterized by cycles of convergence/divergence, then a divergence metric should be able to measure the magnitude and frequency of these cycles. Such metric enables user to monitor convergence/divergence cycles i.e. fix thresholds, control frequency of convergence, and estimate at each time the effort required by the group to converge. In this paper, we define the GroupDiv divergence as the editing distance for a group to the next potential convergence state. This distance is expressed in number of operations to execute by the group. It is possible for each member to know her contribution to this distance, therefore, any member is aware of her own position in the group. Consequently, the proposed metric allows a group to measure its cohesion during collaboration.

Once a group divergence metric is defined, a second issue arises concerning the computation of this metric. Computing group divergence is challenging for several reasons. First, it requires to maintain membership informations. If this is trivial in centralized systems with small groups, such information is more complex to maintain in highly dynamic large groups with no central authority as in distributed version control systems or P2P wikis [15, 16]. Second, Divergence metric computation itself has to be efficient and scalable. If the divergence computation is slow then delivered informations are out-of-date and can be confusing. If divergence computation is not scalable then the quality of delivered information will depend of the size of the group. In this paper, we detailed how it is possible to compute GroupDiv efficiently in a fully decentralized multi-synchronous distributed collaborative system.

This paper contains the following contributions: i) we define a generic formal model for multi-synchronous collaboration ii) we demonstrate how this model can be used to give semantic to existing divergence awareness metrics. iii) we define the GroupDiv divergence metric that is able to quantify divergence, iv) we evaluate GroupDiv divergence with a group of users.

This paper is structured as follows: Section 2 presents background and a motivating example. Section 3 presents related work. Section 4 defines a formal model for multi-synchronous collaboration and defines the group divergence metric. Section **??** details how we used gossiping algorithms to compute a real-time divergence metrics for fully decentralized deployment of multi-synchronous systems. Section **??** details the validation of the proposed algorithm. The last section concludes the paper and points out future work.

2. Background and Motivation

Collaboration can be synchronous, asynchronous or multi-synchronous. Multi-synchronous collaboration is defined by Dourish [1] as:

> Working activities proceed in parallel (multiple streams of activity), during which time the participants are disconnected (divergence occurs); and periodically their individual efforts will be integrated (synchronization) in order to achieve a consistent state and progress the activity of the group.

Multi-synchronous collaboration is widely used because parallel activities can potentially reduce completion time. However when divergence occurs it could generate conflicts where the overhead of solving these conflicts can overwhelm the expected gain [3, 4, 5]. For example, take a scenario of blind modifications [2]: let us consider the collaboration task of editing a book by two users. Each user has a copy that he/she can work on it in isolation. They periodically synchronize and integrate each others' modifications into their own copies. Let us assume that one user is working on a paragraph, while another user is deleting the paragraph. This will waste the work done by the first user, since he/she does not know a priori what the other user is doing.

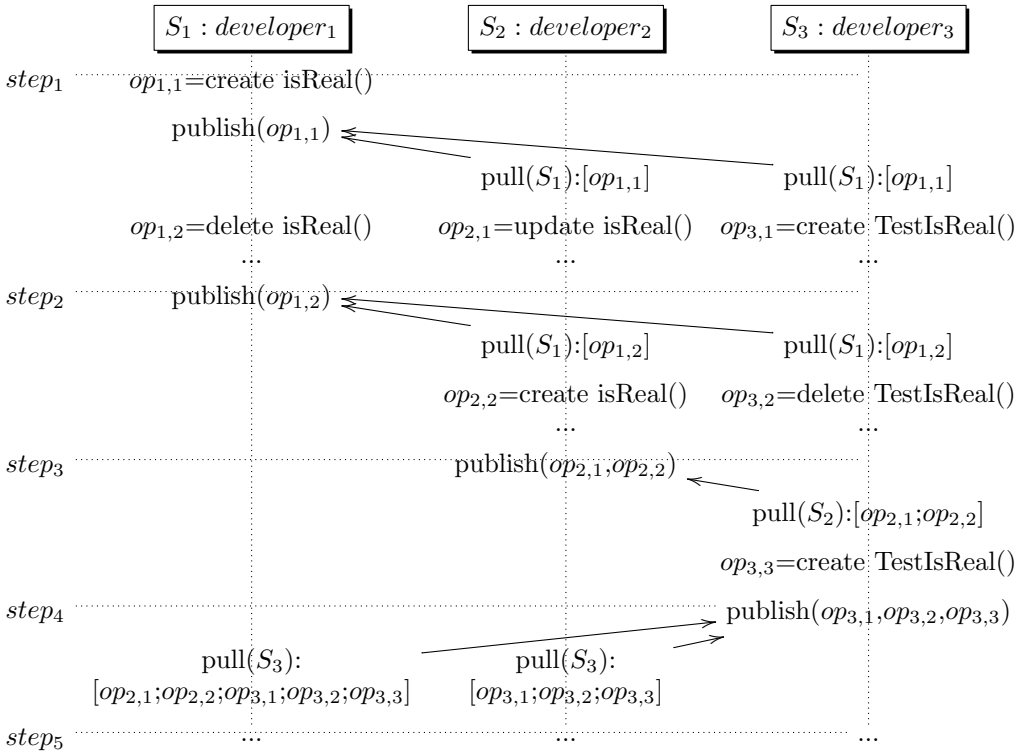

Figure 1: Three developers multi-synchronous collaboration scenario

Several approaches exist to reduce the cost of divergence in multi-synchronous collaborative systems. Planning and coordination can be used to avoid conflicts [17]. By good planning, one can create different parallel tasks that will modify disjoint and independent objects. But fine grained planning can be very costly, and it is not always possible to define disjoint tasks. Furthermore, in the open source development communities, people often collaborate without knowing each other. The development environment is open for any contributor and it is not possible to plan and coordinate in advance. Commit often policy is a best practice for reducing the probability of conflicts [18]. Commit often is not always possible because users commit their modifications after they complete the requested tasks, this means that commit depends on the task completion time. Even if planning and commit policies are good practices, conflicts still exist as detailed in [19, 14]. Zimmermann [19] analyzed CVS repositories, and concluded that, 23% to 47% of all merges had textual conflicts. Brun et al. [14] found that conflicts between developers' copies are rather the norm, they persist on average ten days and they often result in compile, build and test failures.

Planning and coordination can be completed by divergence awareness. Divergence awareness detect divergence and visualize it in order to help users answering the following questions: is there any divergence? With whom? Where? And how much? If existing systems [9, 6, 10, 11, 12, 13, 14] provides some answers to the first three questions, they fail to answer the last one.

Consider a scenario involving three software developers $developer_1$, $developer_2$ and $developer_3$ collaborating on the source code of the same project . This scenario is presented in figure 1 and inspired from [2]. Although at the beginning they divide their work according to predefined tasks, their modifications will overlap later on during their isolated work since their tasks involve some common classes. Each developer works in her/his private workspace respectively S_1, S_2 and S_3. At $step_1$, all workspaces are strictly identical and the group is convergent.

After $step_1$, $developer_1$ adds the method isReal() to the class Integer by generating and applying locally operation $op_{1,1}$. As $op_{1,1}$ is only applied on S_1 divergence exists between S_1, S_2 and S_3, it is localized on

class Integer, and $developer_1$ produced this operation. Then she makes her modifications available for other developers by publishing her operation using $publish(op_{1,1})$. This change nothing for divergence except that now S_2 and S_3 can be aware that a new operation is available from S_1. Consequently, $developer_2$ and $developer_3$ decide to pull $op_{1,1}$ using $pull(S_1)$ and integrate them into their working copy by calling an arbitrary merge tool. At this precise instant, all workspaces should be strictly identical and the global system is convergent again.

Next, $developer_1$ decides to remove the method isReal() from the class Integer. Concurrently, $developer_2$ updates the method isReal() from class Integer such that it returns false instead of real. $developer_3$ tests the class Integer by creating the test class IntegerTest. One of the added methods in that class is the test method for isReal(). Just before $step_2$, three concurrent operations are existing in the system and represent all the divergence present in the system: $op_{1,2}, op_{2,1}, op_{3,1}$

After $step_2$, $developer_1$ publishes $op_{1,2}$ and $developer_2$ and $developer_3$ pull the missing operation and integrate it. $developer_2$ decides to (re)insert the method isReal(), and $developer_3$ decides to remove the test method since the isReal() has been deleted. From the divergence point of view, we can see that $op_{1,1}$ and $op_{1,2}$ are now consumed by all sites and represent some common prefix for 3 sites. Divergence is now represented by unpublished operation of sites S_2 and S_3: $op_{2,1}, op_{2,2}, op_{3,1}, op_{3,2}$

After $step_3$, $developer_2$ publishes his modifications for other developers but only $developer_3$ integrates it and re-create TestIsReal ($op_{3,3}$). At this instant, S_3 has all the seven operations produced in the system, S_2 has just seen four operations, and S_1 two operations.

After $step_4$, $developer_3$ publishes his modifications. $developer_1$ and $developer_2$ pulles from the workspace of $developer_3$ operations done by the later and all preceding operations missing from their respective workspaces. For instance, $developer_1$ will pull the sequence of five operations missing in her workspace. At $step_5$, the whole group is convergent again.

In this scenario, divergence can be seen as an history of conflicts. $developer_1$ deleted the method isReal() while $developer_2$ and $developer_3$ was using it. This can be viewed as a conflict. When they are aware of deletion, they react differently, $developer_2$ reinserts the method while $developer_3$ deletes the test. In fact, after observing the conflict in their own context, each user solved it differently, generating another conflict i.e. a conflict of conflict. When $developer_3$ observed that $developer_2$ reinserted the "isReal method", then he decided to reinsert his test again.

In this scenario, most divergence awareness systems will warn users about concurrent modifications just before $step_2$ as in State Treemap [9]. They will try to estimate size of conflicts as in Palantir [10], and maybe switch to synchronous mode to help solving conflicts as in CollabVS [13] and thus avoid a conflict of conflict.

If a conflict-based approach of divergence makes sense, we think it captures only a part of divergence. Conflicts are characterized by incompatible concurrent write operations that can be performed on the same object or on different object (direct or indirect conflict). However, divergence starts to exist from the moment an operation has been produced on one site and has not been consumed by the others. Therefore, divergence can exist even without any conflict. In the previous scenario, divergence appears just after $step_1$ when $op_{1,1}$ is created and it disappears just after $op_{1,1}$ is consumed by S_2 and S_3. This raises the issue of a formal definition of divergence. Conflict-based approach of divergence leads to define divergence as the lack of convergence, nearly a boolean value. That's why we argue that conflict-based approach of divergences fail to answer the "how much?" question.

To answer this question, we come back on the Dourish definition of Multi-synchronous systems. Dourish suggests the existence of cycles of divergence and convergence within the group. We think that if divergence can be quantified, groups should be able to compute and display these cycles. Once quantified, groups are able to fix divergence threshold and constraints convergence frequency. More important, such divergence metric should be able to maintain group cohesion during activity progress i.e. the ability for a group to see if everyone follows the change at the same speed.

For example, imagine just two people working on a document, but only one is writing, the other is just reading and give feedback by mail (see figure 2). There is no conflicts created by this activity i.e. there is no concurrent operations, however, divergence exists between the workspace of the writer and the workspace of the reader. What can be important in this scenario is to keep divergence under a predefined threshold

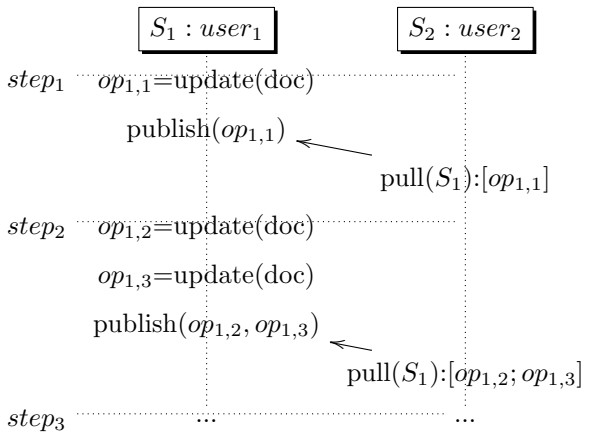

Figure 2: Writer/reader scenario

such as the writer has continuous feedbacks according to her writing speed.

For this purpose, we defined divergence in a multi-synchronous collaborative system as the editing distance of a group to the next potential state of convergence i.e. the minimal number of operations to be performed by the group to reach convergence. We define GroupDiv formally in section 4. A corollary of this definition, is that each group member is able to know how much she contributes to this distance. So it is possible to know for each group member if she/he is walking at the head of the group and if the group is waiting for her/him. Therefore, quantifying divergence allows each group member to adapt her/his speed.

In the previous scenario of writer/reader, the divergence will be the number of operations produced by the writer to integrate in the workspace of the reader. Writer can estimate reader speed by observing evolution of this metric. Reader can also evaluate the writer speed with same observation. In the scenario of figure 1, divergence at $step_2$ should be 6 because each member has to integrate at least two concurrent operations to reach convergence. Before $step_4$, S_1 has to integrate 5 operations, S_2 has to integrate 3 operations, and S_1 is the head. 8 operations has to be integrated by the group to reach the nearest point of convergence and S_1 has the longest path to do.

In the following section, we present related work. Then we present our group divergence metric in sections 4, and its computations is detailed in sections **??** and **??**.

3. Related Work

Existing divergence awareness systems are characterized by divergence metric, when and how they are computed.

Edit Profile [11] makes users aware of "hot areas" and also who is or has been active in various parts of the document. The different contributions of users are quantified and distributed at different levels: document, paragraph, sentence, word and character. It is possible to observe who contributed where, and how much. We compute EditProfile for the scenario in figure 1 considering the extreme case where we have only one document (one node). At $step_2$, Edit Profile will give the following values (2,1,1) for the sites ($S1$, $S2$, $S3$). At $step_3$, the values become (2,2,2), then at $step_4$ (2,2,3), and finally at $step_5$ (2,2,3). This clearly shows that Edit Profile does not give any indication on the group cohesion, nor the work needed in order to achieve convergence. It only gives an indication of the users' past activity.

Concurrency awareness [12] helps users to quickly find where automatic merges have been performed for a P2P network of synchronized wikis and consequently facilitates the verification of final results. Concurrency awareness relies on plausible clocks to detect concurrency *a posteriori*. On the scenario of figure 1, concurrency awareness will warn nothing before $step_2$. It will highlight $op_{1,2}$ and $op_{2,1}$ after $pull(S_1)$ on S_2, and $op_{1,2}$ and $op_{3,1}$ after $pull(S_1)$ on S_3. Concurrency awareness is computed on already integrated

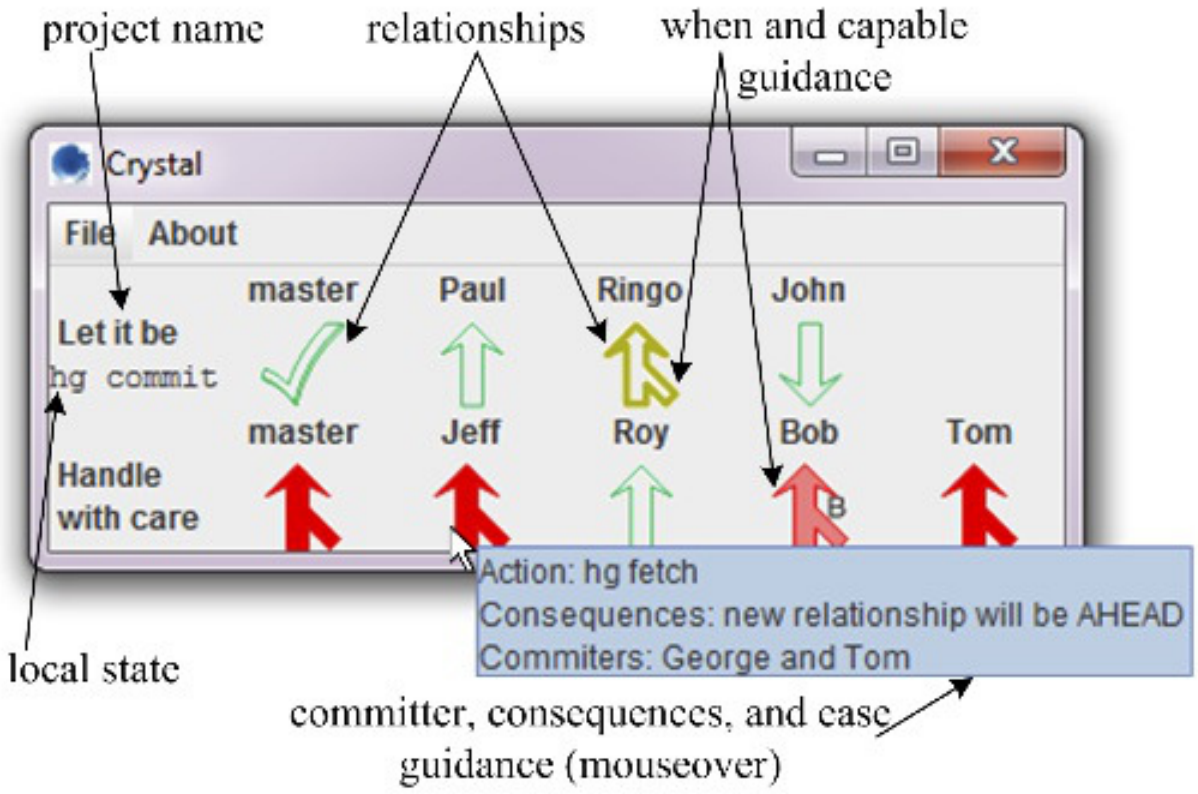

Figure 3: Crystal widget for divergence awareness

operations, so the primary objective is to alert people where concurrency occurred, not where divergence is currently present.

Crystal [14] is a speculative analysis tool that provides concrete information and advice about pending conflicts while remaining largely unobtrusive. It provides developers with information about their development states and the relationships between their repositories and collaborators' repositories. It shows the developer if she is in advance over the other's or if she is behind, which means that modifications have been made to the shared project and she did not consume them yet. It also alerts the developer of the potential conflicts in case she consumes the remote operations (see figure 3). Crystal tracks divergence by checking the state of all workspaces according to a master copy. It helps to locate where divergence is located and with who. For the scenario in figure 1 at $step_1$, Crystal will give the following states (same, same, same) for the three sites ($S1$, $S2$, $S3$) respectively. At $step_2$, the states become (conflict, conflict, conflict). At $step_3$, the states become (behind, conflict, conflict). At $step_4$, the states become (behind, behind, ahead). At $step_5$, the states become (same, same, same).

This clearly shows that divergence is not really quantified, it is impossible to measure the total amount of divergence in the system.

State Treemap [9] shown in figure 4 is an example of divergence awareness widget. It informs participants about the states of shared documents stored in a file system. Different states are defined for a document: *LocallyModified, RemotelyModified, PotentialConflict, etc.* When a document is modified by a participant, it will be marked as *LocallyModified* in her own workspace, while in the others participants' workspaces it will be marked as *RemotelyModified*. Divergence awareness is delivered as a Treemap where each rectangle is colored with the state of the shared object. For instance, if the whole Treemap is white, it means that there is no divergence in the system. If some parts are colored, then users know who changed the file i.e.

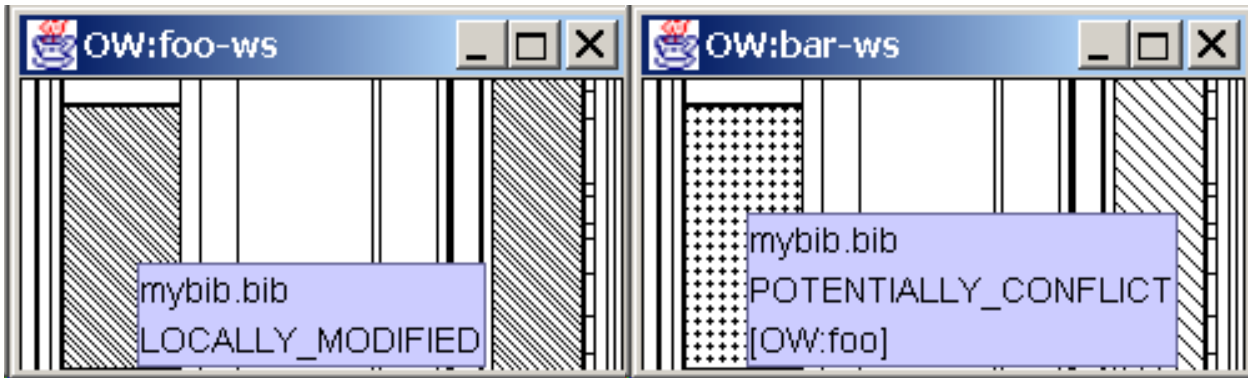

Figure 4: State Treemap

owner "foo" in figure 4. The Treemap itself helps users to know where divergence is located. The number of rectangles of different colors can be seen as a quantification of the divergence in the system.

For the scenario in figure 1 if we consider that we have only one shared object then at $step_1$, State Treemap will give the following states (up-to-date, up-to-date, up-to-date) for the three sites ($S1$, $S2$, $S3$) respectively. At $step_2$, the states become (potential conflict, potential conflict, potential conflict). At $step_3$, the states become (remotely modified, potential conflict, potential conflict). At $step_4$, the states become (remotely modified, remotely modified, locally modified). At $step_5$, the states become (up-to-date, up-to-date, up-to-date).

In State Treemap, different users do not see the same Treemap (see figure 4). So the quantification of divergence as the number of rectangle of different colors will be different. The quantity of divergence in the global system should not depend of the workspace the user is working in, it should estimate the entropy relative to global state of the system with all its workspaces. In this paper, we propose a group divergence metric that aims to quantify divergence as the editing distance to next potential convergence point. Furthermore, State Treemap relies on a central server for performing metrics computations. Users must open a session on this server and send in real-time all local changes in order to get awareness. This kind of architecture can hardly be transposed to more decentralized multi-synchronous collaborative systems such as distributed version control systems or P2P wikis. These systems have been designed to avoid single point of failure, or central authority. They do not maintain membership. Sites just follow the updates of other sites as in social network and generate complex networks of synchronization. In such conditions, how to deliver divergence awareness that needs to build a global knowledge about the system?

Another approach of divergence awareness based on operational transformation [20] is described in [6]. For convenience reason, we will call it OT divergence awareness. In this approach, sites are synchronized on demand but they exchange unpublished operations in real-time as in a distributed real-time editor. An OT algorithm simulates the integration of remote operations in real-time and computes conflict objects. The size of all conflict objects determines the quantity of divergence on each site. Next, this quantification can projected on objects as described in figure 5.

We compute OT divergence at $step_2$ of the scenario presented in figure 1. First $op_{1,2}$ is integrated on S_2 and site S_3. On S_2, $op_{2,1}$ is transformed according to $op_{1,2}$. We suppose that a conflict is detected by transformation function and $op_{2,1}$ is transformed into an operation $op'_{2,1} = conflict(op_{1,2}, op_{2,1})$. On S_3, with the same reasoning, $op_{3,1}$ is transformed into $op'_{3,1} = conflict(op_{1,2}, op_{3,1})$. Finally, $op'_{2,1}$ is integrated on S_3 and $op'_{3,1}$ is transformed into $op''_{3,1} = conflict(conflict(op_{1,2}, op_{2,1}), conflict(op_{1,2}, op_{3,1}))$. At the end of the process, each site will have computed the sequence $[op_{1,2}; op'_{2,1}; op''_{3,1}]$. The divergence in the system is the size of conflict objects produced by this sequence. As the computed sequence is the same for all sites, the OT divergence metric ensures that all sites will see the same value of the metric. OT divergence awareness has been designed for measuring conflicts introduced by concurrent writes. If there

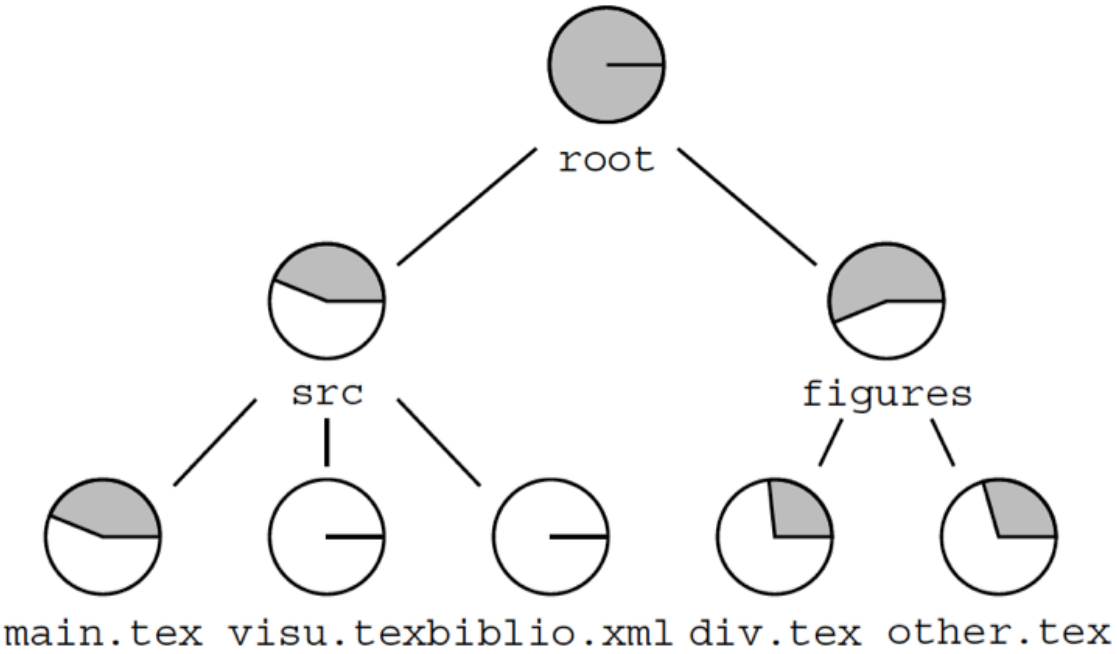

Figure 5: Operational Transformation divergence awareness (OT)

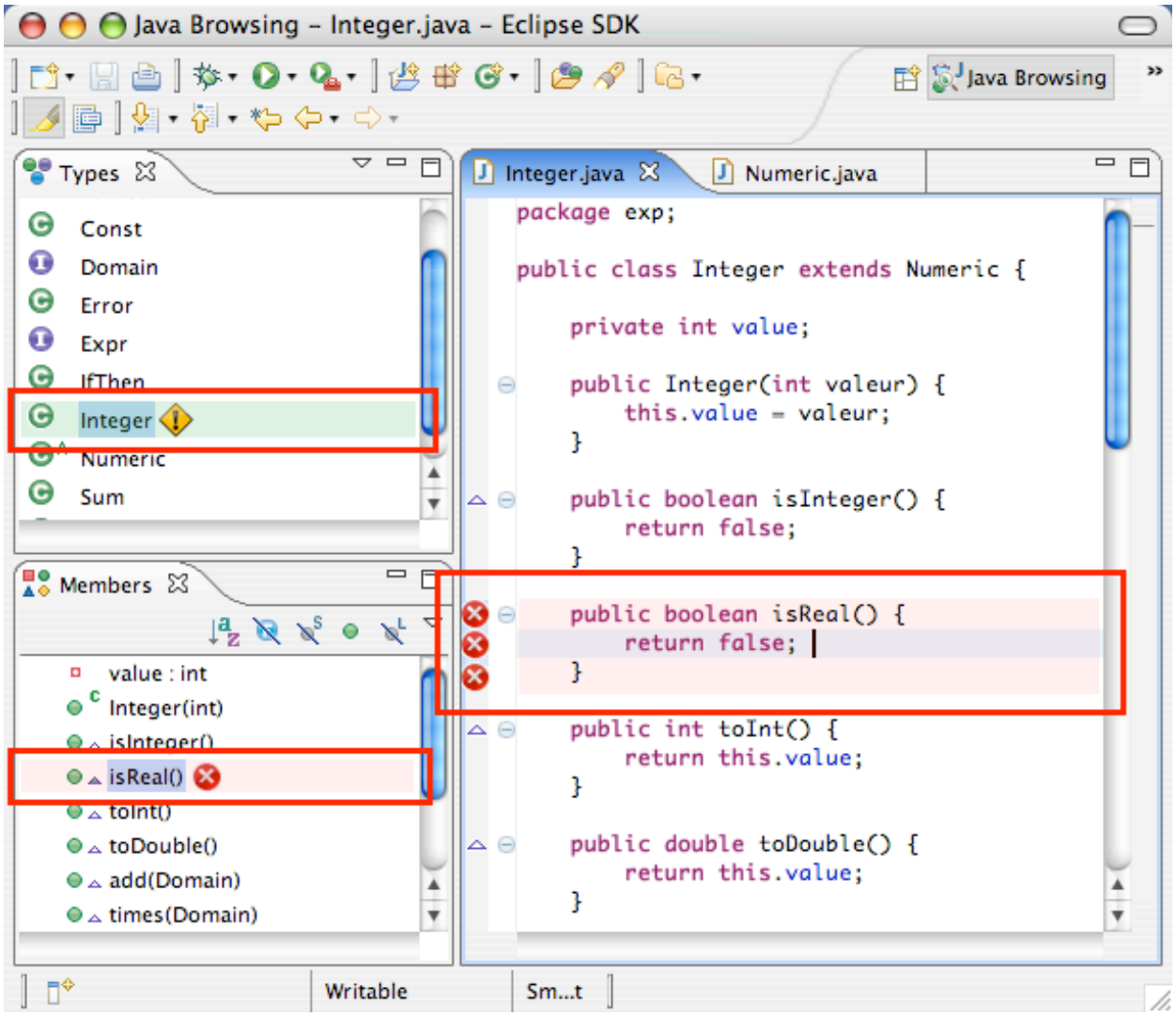

Figure 6: Ghost operations result for the scenario in figure 1

is no concurrent writes, no divergence will be detected. Therefore, OT divergence awareness cannot detect divergence for the writer/reader scenario presented in figure 2.

Ghost operations [2] provides awareness on concurrent ongoing activities to prevent blind modifications while preserving privacy. Ghost operations represent real unpublished operations, some parameters of operations can be blurred according to user preferences to better preserve privacy. Running ghost operations on the scenario presented earlier in figure 1 would give the results shown in figure 6. From this results, we can clearly see that the class *Integer* has blind modifications, and we can also see that these modifications are in the method *isReal*(). Figure 6 shows that ghost operations only notify users where is divergence but it fails answering the question: how much divergence exists in the system?

Palantir [10] is a workspace awareness designed for configuration management systems. It enables early detection of potential conflicts arising from concurrent changes to the same file or dependency violations in ongoing parallel work (see figure 7). For instance in $step_2$ of the motivating scenario of the figure 1, Palantir will detect conflicts between S_1 and S_2 and dependency conflicts between S_1 and S_3. Palantir measures the severity of changes based on ratio of lines changed, added or deleted according to total number of lines. The

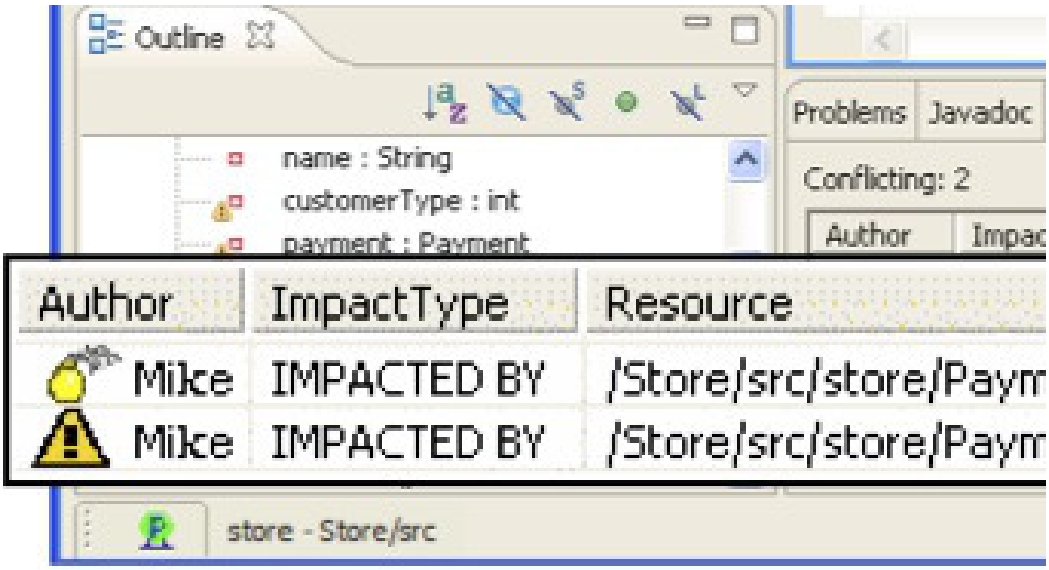

Figure 7: Palantir divergence awareness

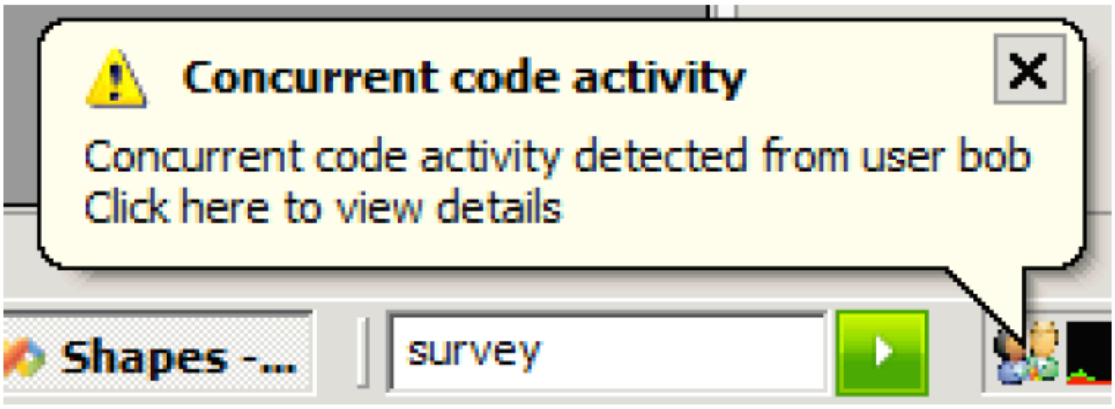

Figure 8: CollabVS Concurrent activity notification

heart of Palantir are the events on ongoing activities in each workspace.

FASTDash [21] is another awareness system designed for small teams of 3-8 collaborating programmers. It provides contextual awareness information such as which code files are changing, who is changing them and how they are being used. This awareness mechanism does not scale for large-scale decentralized systems. Furthermore, this awareness system fails answering the question: how much divergence exists in the system?

CollabVS [13] proposes a semi-synchronous collaboration model in which editing phases are asynchronous and conflict detection and recovery phases can be asynchronous or synchronous. On detecting conflicts, CollabVS displays a notification balloon (see figure 8), user can ignore the notification or go further to have more information about the conflicts and can start a collaborative session with other developers for reviewing and fixing conflicts. CollabVS does not quantify divergence, for instance, in $step_2$ of the motivating scenario of the figure 1, CollabVS will detect conflict between S_1 and S_2 and indirect conflict between S_1 and S_3. CollabVS enables collaborative resolution of conflicts.

All previous divergence awareness systems do not formally define the underlying formula they use to calculate their metrics. Existing divergence awareness systems keep users aware of the presence of divergence,

potential conflicts and where conflicts are located, but they poorly quantify divergence in a multi-synchronous collaborative system and they are not suitable for fully decentralized systems. Our proposal completes the existing divergence metrics with a group divergence metric. This original metric is formally defined and can be efficiently computed in fully decentralized systems.

4. Group Divergence Awareness Formal Model

We observed that existing multi-synchronous collaborative systems behave like optimistic replication systems [22]. An optimistic replication model considers N sites where any kind of objects are replicated. We can say that a site corresponds to a stream of activity in Dourish definition [1]. Objects can be modified anytime, anywhere by applying an update operation locally. According to the optimistic replication models, every operation follows the following lifecycle:

1. An operation is generated in one site, in isolation. It is executed immediately without any locking, even if the local site is off-line. This is the disconnection phase of the multi-synchronous collaboration where divergence is growing.

2. It is broadcasted to all other sites. The broadcast is supposed reliable. All generated operations will eventually arrive to all sites. Pairwise synchronization of sites is a way to broadcast operations to all sites. There is no constraint about how operations are disseminated (broadcast, anti-entropy, pairwise synchronization, gossiping, etc.). We just suppose that a graph of dissemination exists between sites. This graph represents a collaboration network.

3. Received operations are integrated and re-executed. This is the synchronization phase of the multi-synchronous collaboration. Integration relies on merge algorithms such as those used in operational transformation[20]. In this paper, we suppose that the merge algorithm is deterministic, commutative, and associative i.e. merging operations produce the same state in all sites whatever the order of reception of concurrent operations.

Different consistency models can be applied at this stage, causal consistency, eventual consistency, intention preservation etc. We made no hypothesis about the consistency model used. However, divergence awareness relies on concurrent operations analysis and two operations are concurrent if there is no causal relation between them. Causal relations aka "happened-before" relations are defined in [23], we use these definitions for multi-synchronous collaboration systems.

The general idea of the group divergence awareness is:

- Following the optimistic replication model, each site builds a causal history of operations i.e. an history that ensure causal order with causality relation as defined by Lamport [23]
- If all these causal histories are merged, we obtain a maximal causal history, we call it H_{max}.
- If every local causal history is equal to H_{max}, then the convergence is achieved.
- Otherwise, there is a divergence in the system. This divergence is the number of operations to be integrated by the *group* to reach convergence i.e. the number of operations that belongs to H_{max} and not in the local history of all sites. Consequently, our vision of divergence has to be understood as a group divergence and not as an editing distance between two members of this group. This divergence metric makes all members aware of the minimal distance for the group to reach the next potential convergence point represented by H_{max}. It is possible for each member to know how she contributes to this distance, so any member is aware of her own position in the group.
- Computing such divergence metric requires to determine the membership; who is in the group? Answering this question is challenging because multi-synchronous models have no clear procedure for joining and leaving the group. Next, computing the number of operations to be integrated by the group to reach convergence requires a global knowledge of the state of all sites. Membership and global knowledge are not a problem for building a divergence model, however, they are challenging for metric computation at run-time.

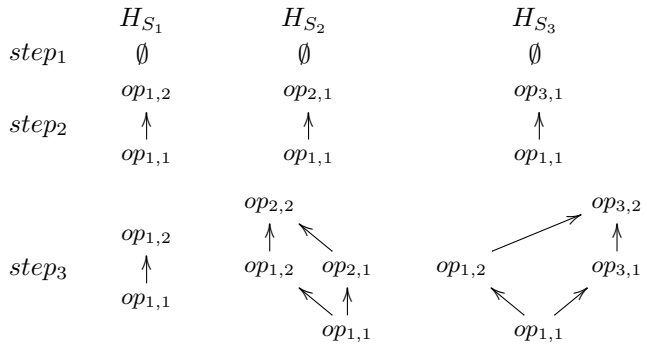

Figure 9: Causal history for three sites at $step_1$, $step_2$ and $step_3$

In the following, we define $\mathbb{S}$ as the set of sites in the system at the divergence awareness computation time, and $N = |\mathbb{S}|$ as the number of sites. Each site is uniquely identified by a unique identifier id, and it maintains a counter c_{id} that increments when the site generates a new operation. An operation op is uniquely identified by the pair (id, c_{id}), where id is the identifier of the generating site and c_{id} is its counter value when it generated the operation. The operation can be annotated with meta-data such as author, timestamp, modified object, position, operation type, etc.

As in [20], following Lamport [23] we define a causal (partial) ordering relation on operations in terms of their generation and execution sequences as follows.

Definition 1 (Causal Ordering Relation "$\rightarrow$")**.** *Given two operations op_a and op_b generated respectively at sites i and j then $op_a \rightarrow op_b$ iff (1) $i = j$ and the generation of op_a* happened before *the generation of op_b; (2) $i \neq j$ and the execution of op_a at j* happened before *the generation of op_b; (3) $\exists op_x$ such that $op_a \rightarrow op_x$ and $op_x \rightarrow op_b$*

For example in the collaboration scenario in figure 1, operations $op_{1,1}$ and $op_{1,2}$ are generated on site S_1 and $op_{1,1}$ is generated before $op_{1,2}$, so we can express that $op_{1,1} \rightarrow op_{1,2}$, $op_{1,1} \rightarrow op_{3,1}$, the execution of $op_{1,1}$ at site S_3 happened before the generation of $op_{3,1}$ at site S_3. Remember according to the optimistic replication model an operation generated on a site is executed immediately.

Definition 2 (Concurrent operations "$\|$")**.** *Given any two operations op_a and op_b, op_a and op_b are said concurrent iff $\neg(op_a \rightarrow op_b) \wedge \neg(op_b \rightarrow op_a)$, which is expressed as $op_a \parallel op_b$.*

For example, $op_{1,2} \parallel op_{2,1}$ and $op_{1,2} \parallel op_{3,1}$ in figure 1 .

We associate a local causal history H_{S_i} to every site S_i in the system, H_{S_i} is defined as follow.

Definition 3 (Local Causal history "H_{S_i}")**.** *Let H_{S_i} the causal history of the site S_i, for any operation $op_b \in H_{S_i}$ if $op_a \rightarrow op_b$ then $op_a \in H_{S_i}$.*

In other words, H_{S_i} corresponds to all operations generated and received by the site S_i. The definition implies that if an operation belongs to the local history, then all operations that causally precede this operation belongs also to the local history. Consequently, the history is complete.

Figure 9 shows the evolution of the causal history of sites S_1, S_2 and S_3 at the different steps of the scenario in the figure 1. The causal history is modeled as a directed acyclic graph with the operations as nodes and causal ordering relations as arcs. For instance, in $step_2$ in $site_3$, $op_{1,1} \rightarrow op_{1,3}$ and $op_{1,2} \parallel op_{3,1}$.

A site has only one function to manipulate its causal history, this function inserts new operation into the causal history. There is no function that deletes an operation from the causal history.

In order to compute the group divergence awareness, we need to calculate the maximal causal history H_{max} of the sites participating in the system at the computation moment. We define H_{max} as follow.

Definition 4 (Maximal causal history "H_{max}").

$$H_{max} = \bigcup_i H_{S_i} : \forall S_i \in \mathbb{S}$$

H_{max} represents the next potential state of the system i.e. as state where the group divergence is null. From H_{max} we can deduce the total number of unique operations in the system.

Definition 5 (Total number of operations "op_{tot}").

$$op_{tot} = |H_{max}|$$

We define the global divergence of a site as the the number of operations in H_{max} that are not in its local causal history.

Definition 6 (A site global divergence "$GD(S_i)$"). *Let $S_i \in \mathbb{S}$, H_{S_i} its local causal history and H_{max} is the maximal causal history in the system then*

$$GD(S_i) = |H_{max} \setminus H_{S_i}|$$

This corresponds to the number of concurrent operations that site S_i has to integrate to reach the next potential convergence state represented by H_{max}.

In the figure 10, we compute the $GD(S_1)$, $GD(S_2)$ and $GD(S_3)$ at each step of the scenario of the motivating example. For instance, at the beginning, site S_1, $GD(S_1) = 0$ therefore S_1 is convergent wrt to S_2 and S_3. In $step_2$, $GD(S_1) = 2$, S_1 needs to integrate two operations to re-establish convergence. In $step_3$, $GD(S_1) = 4$, divergence is increased, in $step_4$, $GD(S_1) = 5$ more divergence, in $step_5$, $GD(S_1) = 0$, convergence is re-established.

Now we can define the group divergence metric, as the total number of operations to be consumed in the system to reach the next potential convergence state. And we note it: GD_{tot}, formally it corresponds to the sum of operations in H_{max} that are not in H_{S_i} of every site S_i in the system.

Definition 7 (Group divergence "GD_{tot}").

$$GD_{tot} = \sum_i |H_{max} \setminus H_{S_i}| : \forall S_i \in \mathbb{S}$$

Figure 10 shows the results of calculating H_{max}, $GD(S_i)$ and the group divergence GD_{tot} for the scenario of figure 1. For example, at $step_4$, every user knows that the group divergence is $GD_{tot} = 8$. This represents the distance for next potential convergence state. Every user knows her own contribution to this distance. S_1 has to integrate five operations: $\{op_{2,1}, op_{2,2}, op_{3,1}, op_{3,2}, op_{3,3}\}$, S_2 has to integrate three operations:$\{op_{3,1}, op_{3,2}, op_{3,3}\}$ and S_3 is up-to-date. Figure 11 shows a possible visualization of GD_{tot} on the scenario of three developers based on results presented in figure 10.

As defined in definition 7, GD_{tot} requires to compute the difference between sets.

However, given $H_{S_i} \subseteq H_{max}$ we can rewrite global divergence $GD(S_i)$ for a site S_i using the total number of operations in the system op_{tot} as follow.

Definition 8 (Site global divergence using aggregation "$GD(S_i)$").

$$op_{tot} = \sum_{i=1}^{N} c_{S_i} : \forall S_i \in \mathbb{S}$$
$$GD(S_i) = op_{tot} - |H_{S_i}|$$

Consequently, we can rewrite the computation of group divergence GD_{tot} in the system with only aggregation functions as follows.

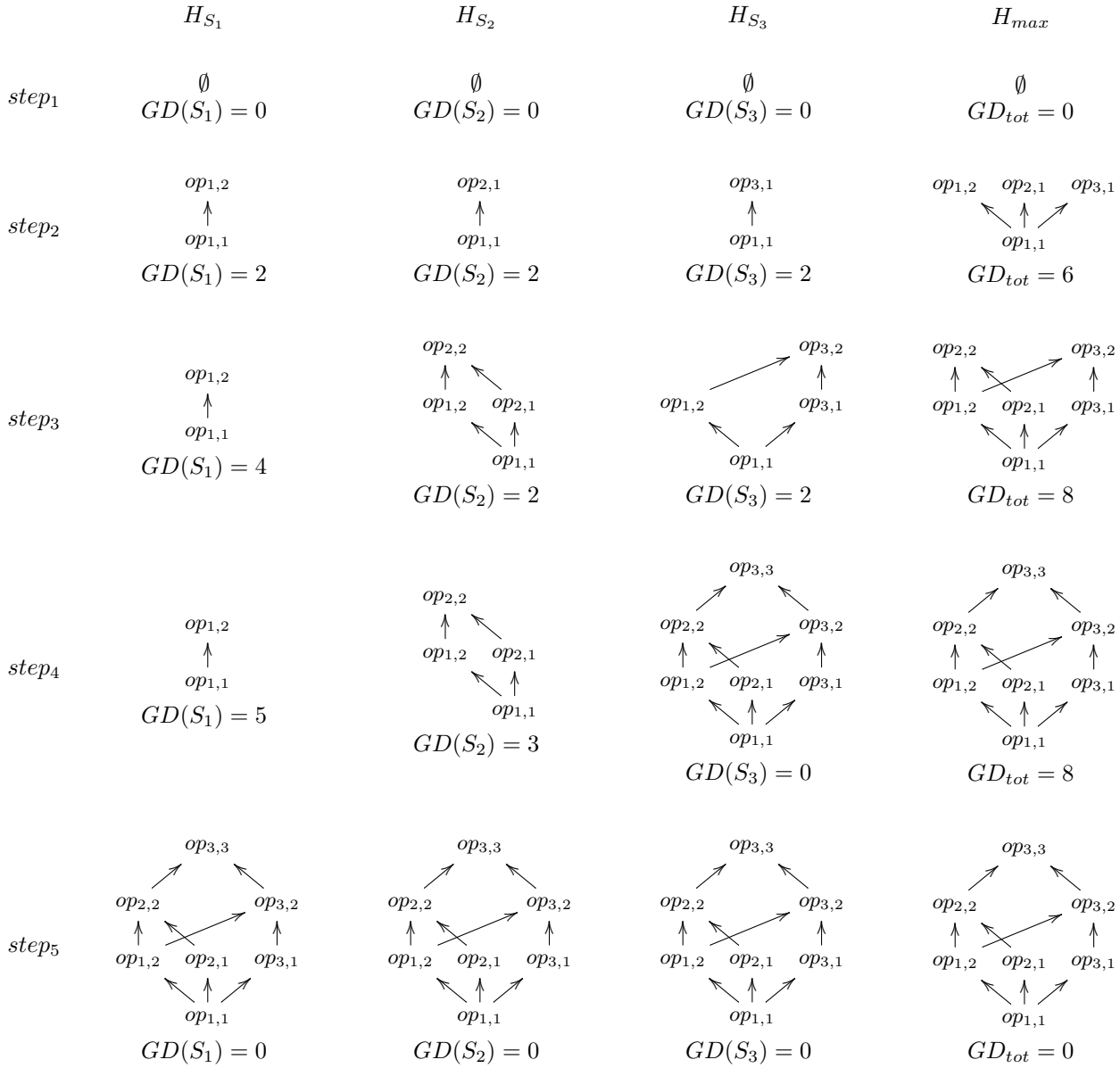

Figure 10: Max causal history and group divergence for three sites

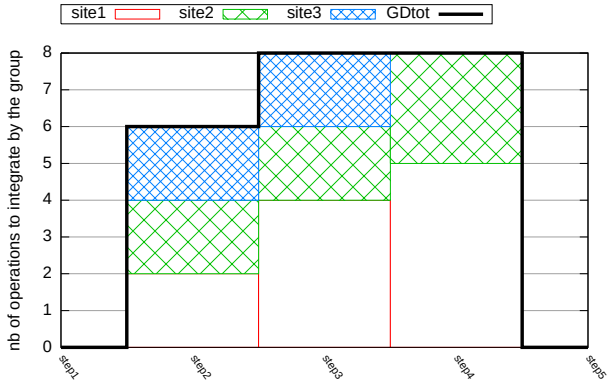

Figure 11: Divergence evolution on scenario of three developers

Step	$\nabla(S_1, S_2)$	$\nabla(S_1, S_3)$	$\nabla(S_2, S_3)$	DIV_∇	GD_{tot}
$step_1$	0	0	0	0	0
$step_2$	2	2	2	6	6
$step_3$	2	2	4	8	8
$step_4$	2	5	3	10	8
$step_5$	0	0	0	0	0

Table 1: Comparing divergence using ∇ to group divergence GD_{tot}

Definition 9 (Group divergence using aggregation 'GD_{tot}")**.**

$$GD_{tot} = op_{tot} \times N - \sum_{i=1}^{N} |H_{S_i}|$$

To better understand GD_{tot} definition, we will compare it with two common strategies for existing divergence awareness metrics. The first one relies on comparing the causal histories of sites pairwise. The second one relies on a copy of reference for comparing the causal histories. Suppose we define divergence between two sites as the number of missing operations in their respective local causal histories. We note divergence between two sites ∇. This corresponds to the editing distance between the two sites.

Definition 10 (Divergence between two sites "∇")**.** $\nabla : \mathbb{S} \times \mathbb{S} \to \mathbb{N}$, $\nabla(S_i, S_j) = |(H_{S_i} \setminus H_{S_j}) \cup (H_{S_j} \setminus H_{S_i})|$

Definition 11 (Group divergence using ∇)**.** $DIV_\nabla = \sum_{i,j} \nabla(S_i, S_j)$

We calculate the divergence between the sites in the scenario in figure 1, the results are shown in the table 1.

In order to calculate the group divergence, it is incorrect to only rely on ∇. As shown in figure 1, at *step*4 S_3 has consumed $op_{2,1}$ and $op_{2,2}$ from S_2. These two operations will be calculated twice: 1) in $\nabla(S_1, S_3)$ and 2) in $\nabla(S_1, S_2)$ this is incorrect.

From this example, we can easily deduce that:

$$GD_{tot} \neq \sum_{i,j} \nabla(S_i, S_j)$$

This illustrates that group divergence awareness cannot rely on distance between sites. GD_{tot} prevents overlapping while counting operations to integrate and keeps the group divergence metric safe.

Step	DIV_{ref}	GD_{tot}
$step_1$	0	0
$step_2$	4	6
$step_3$	4	8
$step_4$	7	8
$step_5$	0	0

Table 2: Comparing divergence using a reference copy DIV_{ref} to group divergence GD_{tot}

State	S_1	S_2	S_3
Locally Modified	false	true	true
Remotely Modified	true	true	true
Potential Conflict	false	true	true
Locally Up-to-date	false	false	false

Table 3: Computing State Treemap for the scenario shown in figure 1 (at $step_4$)

Now we define the divergence based on a copy of reference H_{ref}, as the sum of operations missing from the local site and present in the reference copy, plus the sum of operations missing from the reference copy and present in the local site, we call it DIV_{ref}.

Definition 12 (Group divergence using a reference copy DIV_{ref})**.** $DIV_{ref} = \sum_i |(H_{S_i} \setminus H_{ref}) \cup (H_{ref} \setminus H_{S_i})|$

In order to calculate DIV_{ref} for the scenario in figure 1, we choose the first site's causal history as the reference copy i.e. $H_{ref} = H_{S_1}$ the results are shown in table 2.

We clearly see that DIV_{ref} underestimates the work needed for the group to reach the next potential point of convergence.

Most existing divergence awareness systems use a copy of reference to compute divergence and focus on specific kind of operations in the causal histories i.e. conflicts operations, local operations and remote operations.

Awareness can be interpreted as the projection of H_{max} according to these operations. Therefore, we believe that GroupDiv formalism can be used to define existing divergence awareness.

For instance, we can rewrite State Treemap for one object using GroupDiv model as follows:

Definition 13 (Locally-modified)**.** *The site S_i is in a locally-modified state if* $LM(S_i) = \exists op \in H_{S_i}, \forall S_{j \neq i} \in \mathbb{S} : op \notin H_{S_j}$

Definition 14 (Remotely-modified)**.** *The site S_i is in a remotely modified state if* $RM(S_i) = \exists S_{j \neq i} \in \mathbb{S}, \exists op \in H_{S_j} : op \notin H_{S_i}$

Definition 15 (Potential-conflict)**.** *The site S_i is in a potential-conflict state if* $PC(S_i) = LM(S_i) \wedge RM(S_i)$

This formalization demonstrates that State Treemap only relies on causal histories for computing its states. Suppose, there is only one shared object and we run State Treemap on the scenario shown in figure 1 (at $step_4$). We will obtain:

At the same state, the group divergence metric will give a distance $GD_{tot} = 4 + 2 + 2 = 8$. This clearly demonstrates that State Treemap allows user to perceive divergence and where it is located, but not really to quantify it.

We can also rewrite Palantir states using GroupDiv formal model as follows: If we annotate the operation with a type, then we can check if a document has been created on a site using the operation type *create document*. This corresponds to Palantir populated state. Or we can check if a document has returned to its original state if we find an operation type *undo*. This corresponds to the change reverted state of Palantir.

State	S_1	S_2	S_3
Change in Progress	true	true	true
Change Severity	5	5	5

Table 4: Computing Palantir for the scenario shown in figure 1 (at $step_4$)

Definition 16 (Populated)**.** *A document has been created on a site.* $Pop(S_i) = \exists op \in H_{S_i} : op.type =$ "*createdocument*"

Definition 17 (Change In Progress)**.** *This state is similar to the Locally-Modified or Remotely-Modified states already mentioned in State Treemap.* $CP(S_i) = LM(S_i) \vee RM(S_i)$

Definition 18 (Change Reverted)**.** *The document has returned to its original state.* $CR(S_i) = \exists op \in H_{S_i} :$ $op.type =$ "*undo*"

Definition 19 (Change Severity)**.** *The number of operations that have been done on a document.* $CS(S_i) = |\{op \mid \forall S_{j \neq i} \in \mathbb{S} : (op \in H_i \setminus H_j) \vee (op \in H_j \setminus H_i)\}|$

If we interpret Palantir system with GroupDiv, sharing events containing operations between all participants in real-time is like building H_{max} in each workspace. Next, different projections according to various meta-data can be performed locally such as conflict interpretation and impact analysis. If we compute Palantir states on the scenario shown in figure 1 (at $step_4$). We will obtain:

Compared to GroupDiv, Palantir does not allow to observe the cycle of divergence/convergence, it focuses on estimating size of direct or indirect conflict as in OT divergence awareness [6]. Next, we proposed an efficient algorithm to compute the group metric that does not require to flood the network to build H_{max} in each workspace.

Concurrent awareness [12] can be defined using GroupDiv as follows:

Definition 20 (Concurrent Modification)**.** *The site* S_i *is in a concurrent modification state if* $CM(S_i) = \exists op_1, op_2 \in H_i : op1 \parallel op2$

If we compare with Edit Profile with GroupDiv, Edit Profile can be as the projection of H_{max} according to the structure of the shared document in order to deliver localized awareness.

5. Evaluation

We conducted two-tiered experimentation that evaluated the effectiveness and usability of GroupDiv through a collaborative writing task. We present and analyze our experiment results by addressing three research questions :

1. *D*oes group divergence awareness affect participants behavior ?
2. *D*oes group divergence awareness affect the group cohesion ?
3. *D*oes group divergence awareness affect the time-to-completion for tasks with divergence?

The first question is answered through analysis the actions of participants after a divergence occurrence. The second question is answered through a quantitative analysis of the divergence. If the divergence is under a threshold then the group cohesion is maintained. The third question is answered by comparing task completion time when using GroupDiv or not.

5.1. Experiment Setup

The goal of the experiments was to mimic a collaborative writing of a book with writer and reader roles [**?**]. In this collaborative writing, a participant is responsible of writing a section and another one can review it and and add comments..

We designed a simple text-based experimentation and not a development one to avoid the individual difference in technical skills that could impact the result of the evaluation. The designed experimentation is without conflicts because we want to show that divergence can exist without conflicts and because the impact of the conflicts on parallel activities have studied in several works [**?** **?**].

Two experimentations were conducted. As a multi-synchronous collaborative system we use *g*it. The first experimentation uses *g*it with GroupDiv (called Experimental group) and the second uses *g*it without GroupDiv (called Normal group). A short tutorial about *g*it were given to participants to ensure that they could understand *p*ull and *c*ommit commands of *g*it. A short description for GroupDiv widget was given for the experimental group.

The experiments were conducted at the University of Nantes. All experiment participants were and undergraduate and PhD students in computer sciences. We had twenty participants totally, teen in each group.

5.2. Experiment results

6. Conclusion and Future Work

In this paper, we proposed a clean formal definition of divergence in multi-synchronous collaboration model. This definition overwhelm previous ad-hoc definitions and is not limited to the detection of concurrent conflicting write operations. To the best of our knowledge, GroupDiv is the first metric that allows to quantify divergence in multi-synchronous system opening new opportunities for divergence management: monitoring cycles of convergence/divergence, fix thresholds for divergence, fix frequency for convergence and give awareness about group cohesion during activity progress. We also explained how GroupDiv basic concepts can be used to give clean semantic to existing divergence awareness systems.

Next, we demonstrated how GroupDiv can be efficiently computed even in a fully decentralized multi-synchronous system. The simulation demonstrates that aggregate gossip protocols approximate GroupDiv in few seconds even for a network of 10000 edges.

This work opens several perspectives. First, in this paper we focused on defining and computing the group divergence metric. We have, of course, to experiment how this metric can be understood by users and how users react to divergence in usage studies. Second, GroupDiv is built on causal histories where only write operations are present. Somehow, GroupDiv deduces read operations by comparing local histories. We think that integrating read operations directly in the causal history can improve GroupDiv model. Third, we used an overlay network for membership and metrics computations. We think the overlay network can be used to fully host the multi-synchronous system itself and consequently propose a new kind of architecture of multi-synchronous distributed systems.

References

1. Dourish, P.. The Parting of the Ways: Divergence, Data Management and Collaborative Work. In: *4th European Conference on Computer Supported Cooperative Work*. Kluwer Academic Publishers; 1995, p. 215–230.
2. Ignat, C.L., Oster, G., Molli, P., Skaf-Molli, H.. A Collaborative Writing Mode for Avoiding Blind Modifications. In: *Ninth International Workshop on Collaborative Editing Systems, GROUP'07*. 2007, p. 1–6.
3. de Souza, C., Redmiles, D., Dourish, P.. Breaking the code, moving between private and public work in collaborative software development. In: *Proceedings of the 2003 International ACM SIGGROUP conference on Supporting group work*. ACM. ISBN 1-58113-693-5; 2003, p. 105–114.
4. Perry, D., Siy, H., Votta, L.. Parallel changes in large-scale software development: an observational case study. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 2001;**10**(3):308–337.
5. Prudêncio, J.a.G., Murta, L., Werner, C., Cepêda, R.. To lock, or not to lock: That is the question. *Journal of Systems and Software* 2012;**85**(2):277–289.
6. Molli, P., Skaf-Molli, H., Oster, G.. Divergence awareness for virtual team through the web. In: *Integrated Design and Process Technology, IDPT 2002*. Pasadena, CA, USA: Society for Design and Process Science; 2002, p. 1–10.

7. Gutwin, C., Greenberg, S.. Effects of awareness support on groupware usability. In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM Press/Addison-Wesley Publishing Co. ISBN 0-201-30987-4; 1998, p. 511–518.
8. Gross, T., Stary, C., Totter, A.. User-centered awareness in computer-supported cooperative work-systems: Structured embedding of findings from social sciences. *International Journal of Human-Computer Interaction* 2005;**18**(3):323–360.
9. Molli, P., Skaf-Molli, H., Bouthier, C.. State Treemap: an awareness widget for multi-synchronous groupware. In: *Seventh International Workshop on Groupware - CRIWG*. IEEE Computer Society. ISBN 0-7695-1351-4; 2001, p. 106–114.
10. Sarma, A., Redmiles, D., der Hoek, A.V.. Palantir: Early Detection of Development Conflicts Arising from Parallel Code Changes. *IEEE Transactions on Software Engineering* 2011;**99**.
11. Papadopoulou, S., Ignat, C., Oster, G., Norrie, M.. Increasing Awareness in Collaborative Authoring through Edit Profiling. In: *IEEE Conference on Collaborative Computing: Networking, Applications and Worksharing - CollaborateCom 2006*. ISBN 1-4244-0429-0; 2006, p. 1–9.
12. Alshattnawi, S., Canals, G., Molli, P.. Concurrency awareness in a P2P wiki system. In: *International Symposium on Collaborative Technologies and Systems*. IEEE. ISBN 978-1-4244-2248-7; 2008, p. 285–294.
13. Dewan, P., Hegde, R.. Semi-synchronous conflict detection and resolution in asynchronous software development. In: *The Sixth European Conference on Computer-Supported Cooperative Work, ECSCW2007*. Springer; 2007, p. 159–178.
14. Brun, Y., Holmes, R., Ernst, M.D., Notkin, D.. Proactive detection of collaboration conflicts. In: *ESEC/FSE 2011: The 8th joint meeting of the European Software Engineering Conference (ESEC) and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE)*. 2011, p. 168–178.
15. Rahhal, C., Skaf-Molli, H., Molli, P., Weiss, S.. Multi-synchronous Collaborative Semantic Wikis. In: *10th International Conference on Web Information Systems Engineering - WISE '09*; vol. 5802 of *LNCS*. Springer; 2009, p. 115–129.
16. Aslan, K., Skaf-molli, H., Molli, P.. Connecting Distributed Version Control Systems Communities to Linked Open Data. In: *The 2012 International Conference on Collaboration Technologies and Systems (CTS 2012)*. Denver, Colorado, USA; 2012, .
17. Estublier, J., Garcia, S.. Process model and awareness in SCM. In: *Proceedings of the 12th international workshop on Software configuration management*; vol. 12. ACM; 2005, p. 59–74.
18. Wloka, J., Ryder, B., Tip, F., Ren, X.. Safe-commit analysis to facilitate team software development. In: *Proceedings of the 31st International Conference on Software Engineering*; ICSE '09. Washington, DC, USA: IEEE Computer Society. ISBN 978-1-4244-3453-4; 2009, p. 507–517.
19. Zimmermann, T.. Mining Workspace Updates in CVS. In: *Proceedings of the Fourth International Workshop on Mining Software Repositories, ICSE Workshops MSR'07*. 2007, p. 11–11.
20. Sun, C., Jia, X., Zhang, Y., Yang, Y., Chen, D.. Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. *ACM Transactions on Computer-Human Interaction (TOCHI)* 1998;**5**(1):63–108.
21. Biehl, J.T., Czerwinski, M., Smith, G., Robertson, G.G.. Fastdash: a visual dashboard for fostering awareness in software teams. In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM. ISBN 978-1-59593-593-9; 2007, p. 1313–1322.
22. Saito, Y., Shapiro, M.. Optimistic replication. *ACM Computing Surveys* 2005;**37**(1):42–81.
23. Lamport, L.. Times, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM* 1978; **21**(7):558–565.
24. Hartig, O., Bizer, C., Freytag, J.C.. Executing SPARQL Queries over the Web of Linked Data. In: *International Semantic Web Conference*. Springer Berlin, Heidelberg. ISBN 978-3-642-04929-3; 2009, p. 293–309.
25. Le Merrer, E., Straub, G.. Distributed Overlay Maintenance with Application to Data Consistency. In: *Globe 2011, 4th International Conference on Data Management in Grid and P2P Systems*. ISBN 978-3-642-22946-6; 2011, p. 25–36.
26. Ganesh, A.J., Kermarrec, A.M., Massoulié, L.. Peer-to-peer membership management for gossip-based protocols. *IEEE Trans Comput* 2003;**52**(2):139–149.
27. Kempe, D., Dobra, A., Gehrke, J.. Gossip-based computation of aggregate information. In: *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*; FOCS '03. Washington, DC, USA: IEEE Computer Society. ISBN 0-7695-2040-5; 2003, p. 482–491.
28. Montresor, A., Jelasity, M.. PeerSim: A scalable P2P simulator. In: *2009 IEEE Ninth International Conference on Peer-to-Peer Computing*; 214412. Ieee. ISBN 978-1-4244-5066-4; 2009, p. 99–100.
29. Erdos, P., Rényi, A.. *On the evolution of random graphs*. Akad. Kiadó; 1960.
30. Albert, R., Barabasi, A.. Statistical mechanics of complex networks. *Reviews of modern physics* 2002;**74**(1):47.