Divergence Awareness in Distributed Multi-Synchronous Collaborative Systems

Khaled Aslan-Almoubayed

l'université nantes angers le mans
PÔLE DE RECHERCHE ET D'ENSEIGNEMENT SUPÉRIEUR

UNIVERSITÉ DE NANTES

Thèse de Doctorat

Khaled Aslan Almoubayed

Mémoire présenté en vue de l'obtention du
grade de Docteur de l'Université de Nantes
sous le label de l'Université de Nantes Angers Le Mans

Discipline : Informatique
Spécialité : Informatique
Laboratoire : Laboratoire d'informatique de Nantes-Atlantique (LINA)

Soutenue le 26 novembre 2012

École doctorale : 503 (STIM)
Thèse n° : 000000000

Divergence Awareness in Distributed Multi-Synchronous Collaborative Systems

JURY

Rapporteurs :	**M. Bruno Defude**, Professeur, TELECOM SudParis **M. Nicolas Roussel**, Directeur de Recherche, INRIA Lille
Examinateurs :	**M. Marc Gelgon**, Professeur, Université de Nantes **M. Gérald Oster**, Maître de Conferences, Université de Lorraine
Directeur de thèse :	**M. Pascal Molli**, Professeur, Université de Nantes
Co-directrice de thèse :	**M^{me} Hala Skaf-Molli**, Maître de Conferences, Université de Nantes

Contents

List of Figures

List of Tables

1

Introduction

Collaborative systems allow people distributed in time and space to achieve common goals. The Computer Supported Cooperative Work (CSCW) Matrix introduced by Johansen [47] classified collaborative systems according to time and space dimensions. Collaboration can be synchronous i.e. at the same time either in the same physical space, such as; meeting rooms, conference rooms or common workspaces; or in distant spaces such as; videoconference rooms, collaborative editors or shared whiteboards. Collaboration can be asynchronous i.e. at different time either in the same place or in different places.

Multi-synchronous collaboration is another collaboration model that can not fit in the Matrix [78]. According to Dourish [23], multi-synchronous collaboration is defined 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.

In contrast to synchronous and asynchronous collaboration, multi-synchronous collaboration does not give the illusion of single, global stream of activity over the shared data. Muti-synchronous collaborative systems support parallel stream of activities on replicated data. In multi-synchronous collaborative system, it is possible for participants to work at the same time, in the same place but on different copies. In such case, the interaction is not synchronous.

When working disconnected, streams have different views of the data, divergence occurs. Participants have to resynchronize their copies at reconnection to reach a consistent state. Collaboration goes through cycles of convergence and divergence.

All collaborative systems with replication and synchronization functionalities are multi-synchronous collaborative systems. Examples include Distributed Version Control Systems (DVCS) [2] with git [1], Mercurial [2] and

[1] http://git-scm.com/

[2] http://www.selenic.com/mercurial/

Bazaar [3], synchronization tools such as Dropbox [4] or GoogleDrive, collaborative editors such as Distributed Wikis [77] or GoogleDoc.

Multi-synchronous collaboration model can potentially reduce completion time. However, solving conflicts generated by merging concurrent work can overwhelm the expected gain [17, 64, 66].

Different approaches exist to manage divergence. *Planning and coordination* can be used to avoid conflicts [20, 28] but fine grained planning can be very costly, and it is not always possible to define disjoint tasks. *Divergence awareness* is another possible solution to limit divergence and reduce conflicts in multi-synchronous collaboration. Awareness allows participants to establish a mutual understanding to perform joint actions. In multi-synchronous collaboration, divergence is explicit, participants work isolated on their copies and synchronize from time to time. Without awareness, they generate blind modifications [46] leading later to synchronization conflicts and more work to solve these conflicts.

Divergence awareness is important in multi-synchronous collaboration, it can reduce conflicts and allows to avoid blind modifications. Divergence awareness [58] makes participants aware of the quantity and the location of divergence in shared objects. Participants are informed about potential risk of future conflicts. Divergence awareness answers the following questions: is there any divergence? With whom? Where? And how much? Divergence awareness is an implicit coordination mechanism [39, 34], it incites participants to coordinate their actions to reduce divergence.

1.1 Topic and Motivation of this Thesis

In this thesis, we are interested in divergence awareness in decentralized multi-synchronous collaborative systems. Multi-synchronous collaboration allows people to work concurrently on copies of a shared document which generates divergence. Divergence awareness allows to localize where divergence and estimate how much divergence exists among the copies.

Divergence awareness has been provided by different systems, relying on different metrics with different ad-hoc visualizations like: State Treemap [57], Operational Transformation Divergence [58], Palantir [73], Edit Profile [63], Concurrent modifications [3], and Crystal [15].

However, existing divergence awareness metrics are highly coupled to their original applications and can not be used outside their original scope. *Is it possible to define divergence metrics in an abstract multi-synchronous collaboration model?* This allows to reuse, compare and maybe combine metrics.

Divergence metrics are computed for a group of participants. This group has to be defined and maintained through membership. However, decentralized multi-synchronous collaboration model is an editing social network where each participants follows the changes of others. This collaboration model does not provide a membership functionality and hides social relations. Therefore, it is not possible to navigate in the social network. Maintaining group membership in a distributed collaborative system is not easy. Decentralized

[3] http://bazaar.canonical.com/
[4] http://www.dropbox.com

multi-synchronous collaborative systems provide collaboration services without a dedicated service provider. Therefore, there is no central point with a global vision of the systems. Each participant in the system has only a local view of the social network. *How to manage group membership and to discover the collaboration network in decentralized multi-synchronous collaborative system?*

Another topic is related to the interpretation of the computed metrics. Different approaches exist for computing divergence. This can be done through the estimating of the size of conflicts [58], estimating the difference between users' copies and a reference copy [57], or estimating divergence according to multiple copies of reference [15]. Most of these metrics estimate some editing distance between users workspaces and a copy of reference. They do not really try to quantify the divergence for the whole of the group without a copy of reference. *Is it possible to define group divergence? How this group divergence can be computed in a decentralized multi-synchronous collaborative system?* Computing this global metric in a real-time is very challenging since it requires a distributed computation with all related distributed problems such as scalability, availability and privacy preservation.

1.2 Contributions of this Thesis

We propose a formal model to define divergence in an abstract multi-synchronous collaboration model and to compute group divergence awareness in a fully distributed collaborative system. Before proposing this model, we started by studying and analyzing existing divergence awareness in multi-synchronous distributed collaborative systems.

We observed that all existing distributed multi-synchronous collaborative systems rely on the same concepts of sharing causal history [67]. The editing social network is built by exchanging fragments of causal histories among participants. In spite of this, by analyzing existing systems, we observed that these systems define their own divergence metrics without a common formal definition. These metrics are coupled with their applications and cannot be used outside their original scope. The contributions of this thesis are :

1. The first contribution is the definition of the SCHO ontology; a unified formal ontology for constructing and sharing the causal history in a distributed collaborative system. We redefined existing divergence metrics in a declarative way based on this ontology and validate this contribution by using real data extracted from software engineering development projects.

2. The second contribution proposes a membership service for collaborative systems. We extend the SCHO ontology in order to link participants and objects by using FOAF/DOAP vocabularies to be part of linked data initiative. Participants can extract informations from their local workspace and generate RDF datasets. Therefore, each participant can run queries to discover the collaboration social network using Link Traversal Based Query Execution. The advantage of this approach that it does not require a beforehand knowledge of the sites that it will seek for getting the

data to evaluate the query result, which is compatible with the multi-synchronous collaboration model.

3. The third contribution is a formal definition for multi-synchronous collaborative system based on causal histories. We also propose an original group divergence metric that computes global metric for a group. It is the number of operations to integrate by the group to reach a convergence state. This metric allows each group member to know exactly her distance to the next convergence point.

4. As a fourth contribution, we propose an algorithm to compute the group divergence metric for causal histories. The metric is expressed as semantic queries thanks to the SCHO ontology. We validate this algorithm by computing group divergence metric on real histories extracted from different distributed version control systems.

5. As a fifth contribution, we propose a distributed algorithm to compute group divergence metric efficiently using gossiping protocols in a fully decentralized network. We validate the distributed computation of divergence metrics with simulations on a peer-to-peer network.

1.3 Outline of this Thesis

In the following, we detail the structure of this thesis :

Chapter 2: Background. This chapter gives definitions and illustrations of multi-synchronous collaboration model. Next, it describes existing divergence awareness systems and raises the issues that are handled in this thesis.

Chapter 3: SCHO: Shared Causal History Ontology. In this chapter we introduce our formal ontology for sharing causal histories. We use semantic web technologies to define the SCHO ontology for constructing and sharing the causal history in a distributed collaborative system. Then, we define the existing divergence metrics in a declarative way as semantic queries over this ontology. Finally, we validate our approach using real data extracted from software engineering development projects.

Chapter 4: Network Discovery. In this chapter, we introduce the membership problem of multi-synchronous collaborative systems. Decentralized multi-synchronous collaboration model does not define membership service, this is an obstacle to compute divergence metrics in a distributed collaborative systems. Then we show how we can use semantic web technologies and transform any multi-synchronous collaboration system into a social semantic web tool. In these systems, participants can perform semantic queries using Link Traversal Based Query Execution to navigate through the social network. Finally, we run simulations to evaluate the approach, the results show an extremely poor performance, this approach does not scale and makes it impossible to rely on it for computing divergence awareness.

Chapter 5: GroupDiv: Group Divergence Awareness Formal Model. In this chapter, we propose a formal model for multi-synchronous collaborative systems and we define an original group divergence metric. This metric addresses specifically the "how much?" question. It makes users aware of the distance of the group to the next potential convergence point. Then, we propose an algorithm to compute group divergence metric on logs and validates the algorithm with real data from different development projects. Finally, we propose an original approach to compute group divergence metric in real-time in a fully decentralized network and validates the approach with simulations.

Chapter 6: Conclusion and Perspectives. This chapter concludes the thesis and provides related future research lines.

1.4 Publications

This thesis is based on the following publications:

1. *Khaled Aslan, Nagham Alhadad, Hala Skaf-Molli, and Pascal Molli. SCHO: An Ontology Based Model for Computing Divergence Awareness in Distributed Collaborative Systems. In The Twelfth European Conference on Computer-Supported Cooperative Work (ECSCW2011), September, 2011, Aarhus, Denmark.*

Multi-synchronous collaboration allows people to work concurrently on copies of a shared document which generates divergence. Divergence awareness allows to localize where divergence is located and estimate how much divergence exists among the copies. Existing divergence awareness metrics are highly coupled to their original applications and can not be used outside their original scope. In this paper, we propose the SCHO ontology: a unified formal ontology for constructing and sharing the causal history in a distributed collaborative system. Then we define the existing divergence metrics in a declarative way based on this model. We validate our work using real data extracted from software engineering development projects.

2. *Khaled Aslan, Hala Skaf-Molli, and Pascal Molli. From Causal History to Social Network in Distributed Social Semantic Software. In Web Science Conference: Extending the Frontiers of Society On-Line (WebSci2010), 2010, Raleigh, North Carolina, USA.*

Web 2.0 raises the importance of collaboration powered by social software. Social software clearly illustrated how it is possible to convert a community of strangers into a community of collaborators producing all together valuable content. However, collaboration is currently supported by collaboration providers such as Google, Yahoo, etc. following "Collaboration as a Service (CaaS)" approach. This approach raises privacy and censorship issues. Users have to trust CaaS providers for both security of hosted data and usage of collected data. Alternative approaches including private peer-to-peer networks, friend-to-friend networks, distributed version control systems and distributed peer-to-peer groupware; support collaboration without requiring a collaboration provider. Collaboration is powered with the resources provided by the users. If it is easy for a collaboration provider to extract the complete social network graph from the observed interactions. Obtaining social network informations in the distributed approach is more challenging. In fact, the

distributed approach is designed to protect privacy of users and thus makes extracting the whole social network difficult. In this paper, we show how it is possible to compute a local view of the social network on each site in a distributed collaborative system approach.

3. *Khaled Aslan, Hala Skaf-Molli, and Pascal Molli. Connecting Distributed Version Control Systems Communities to Linked Open Data. In Proceedings of the 2012 International Conference on Collaboration Technologies and Systems (CTS 2012), 2012, Denver, Colorado, USA. (Nominated for outstanding paper award).*

Distributed Version Control Systems (DVCS) such as git or Mercurial allow community of developers to coordinate and maintain well known software such as Linux operating system or Firefox web browser. The Push-Pull-Clone (PPC) collaboration model used in DVCS generates PPC social network where DVCS repositories are linked by push/pull relations. Unfortunately, DVCS tools poorly interoperate and are not navigable. The first issue prevents the development of generic tools and the second one prevents network analysis. In this paper, we propose to reuse semantic web technologies to transform any DVCS system into a social semantic web one. To achieve this objective, we propose to extend the *SCHO* ontology. This ontology allows each node of the PPC social network to publish semantic datasets. Next, these semantic datasets can be queried with link transversal based query execution for metrics computation and PPC social network discovery. We experimented PPC network discovery and divergence metrics on real data from some representative projects managed by different DVCS tools.

4. *Khaled Aslan, Hala Skaf-Molli, and Pascal Molli. GroupDiv: Formalizing and Computing Group Divergence Awareness in Multi-Synchronous Distributed Collaborative Systems. In Future Generation Computer Systems (FGCS). Special Issue on Advances in Computer Supported Collaboration Technologies and Systems, Vol. XXX, pp. xxx-xxx, 2012 (Submitted - major revision).*

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. Working in parallel can potentially reduce completion time, however, it introduces blind modifications and the overhead of solving conflicts introduced by concurrent modifications can overwhelm the expected gain. Divergence awareness quantifies divergence and answers the following questions: is there any divergence? With whom? Where? And how much? This paper presents a generic formal model for defining divergence metrics and demonstrates how existing metrics can be expressed in this model. It proposes also an efficient algorithm to compute divergence metrics in a fully decentralized network and validate through simulations.

5. *Khaled Aslan, Pascal Molli, Hala Skaf-Molli, and Stephane Weiss. C-Set: a Commutative Replicated Data Type for Semantic Stores. In Fourth International Workshop on Resource Discovery (RED) 2011, Heraklion, Greece.*

Web 2.0 tools are currently evolving to embrace semantic web technologies. Blogs, CMS, Wikis, social networks and real-time notifications; integrate ways to provide semantic annotations and therefore contribute to the linked data,

and more generally to the semantic web vision. This evolution generates a lot of semantic datasets of different qualities, different trust levels and partially replicated. This raises the issue of managing the consistency among these replicas. This issue is challenging because semantic data-spaces can be very large, they can be managed by autonomous participants and the number of replicas is unknown. A new class of algorithms called Commutative Replicated Data Type are emerging for ensuring eventual consistency of highly dynamic content on P2P networks. In this paper, we define C-Set a CRDT specifically designed to be integrated in Triple-stores. C-Set allows efficient P2P synchronization of an arbitrary number of autonomous semantic stores.

2

Background

Contents

A widely accepted definition of collaborative systems (also refereed as Groupware) is the definition given by Ellis [26]: *Computer-based systems that support groups of people engaged in a common task (or goal) and that provide an interface to a shared environment.*

Many collaborative systems exist such as decision rooms, group calendars, video conferences, workflow and collaborative editors. These systems have different functions and different synchronization periods. The collaboration in these systems can be synchronous, asynchronous or multi-synchronous. According to Ellis et al. [26], synchronous interaction means same time or real-time interaction and asynchronous interaction means different time or not real-time interaction. The Computer Supported Cooperative Work (CSCW) Matrix introduced by Johansen [47] classifies the collaborative system according to the time and place dimensions as shown in the figure 2.1. The place axis considers the spatial distance between participants and the time axis considers the temporal distance.

- The time axis is divided as follows:

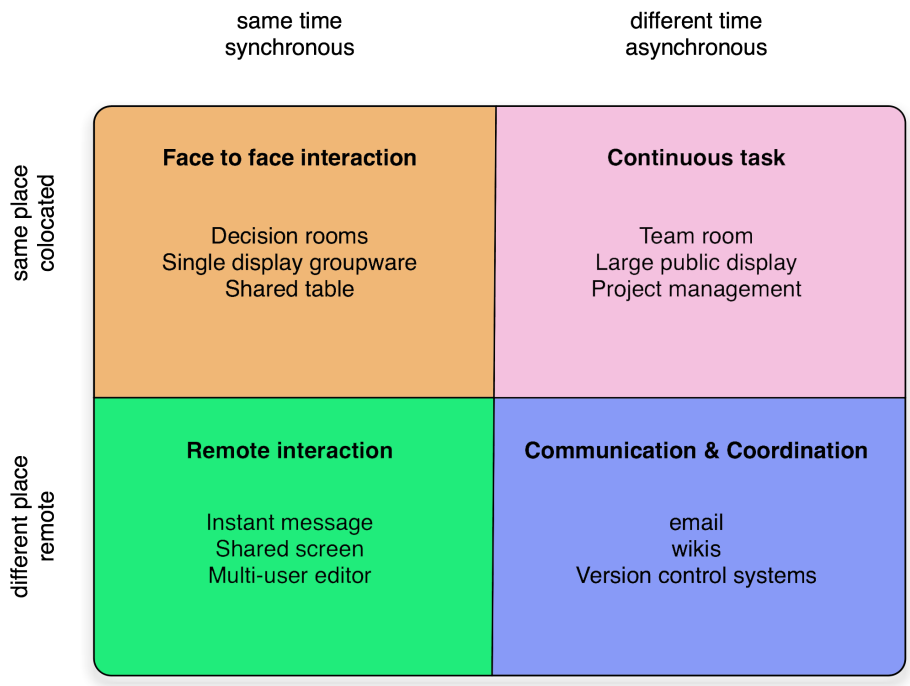


Figure 2.1: The CSCW matrix

1. Same time or synchronous: This corresponds to the situation where group members interact with the collaborative system simultaneously.
2. Different time or asynchronous: This corresponds to the situation where group members interact with collaborative system at different moments.

- The place axis is divided as follows:

1. Same place: This corresponds to the situation where group members are co-located in the same place when they interact with the collaborative system.

(a) Synchronous collaboration using Microsoft surface 2.0

(b) Asynchronous collaboration in NASA Control Room

Figure 2.2: Synchronous/asynchronous same place collaborative systems

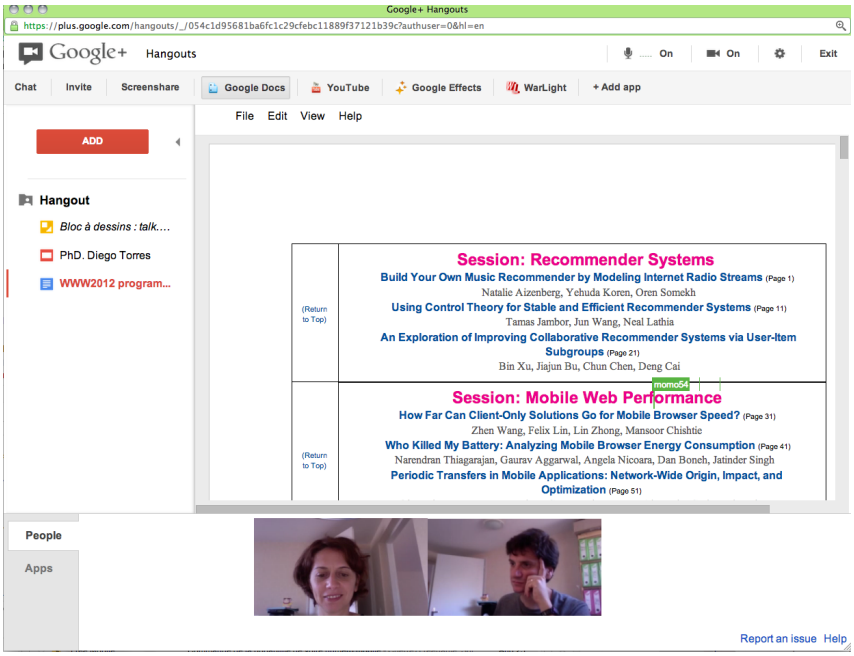


Figure 2.3: Synchronous collaboration in different place with hangout of GooglePlus

2. Different place: This corresponds to the situation where group members are geographically distributed.

This matrix allows to classify the collaborative systems to: co-located or distributed, synchronous or asynchronous. Four different regions are represented by the matrix:

1. "same time" and "same place", as in a face to face synchronous interaction with single display groupware [80]. In a single display groupware, users interact through a shared screen. For instance, the figure 2.2a shows the touch table of Microsoft which allows different persons at the same place to interact together.

2. "different time" and "same place", here we find asynchronous interaction in the same place using a large public display [60] or a control room. Figure 2.2b shows the NASA mission control center [1] where a room has different computers connected to large displays, this allows the continuous task control. The idea is to allow people in the room to control the applications projected in the room at different time.

3. "same time" and "different place", this region is characterized by geographical distribution, participants are located in different places (in different offices, different places in the same city, different cities or different countries) but they interact with the system at the same time. Examples include Skype, and hangout of GooglePlus, where people can be in different places, they communicate synchronously using video and voice. Many real-time collaborative editors such as Grove [26] and GroupKit[70]

[1] http://en.wikipedia.org/wiki/Control_room

are classified in this region. Figure 2.3 shows a video/audio conference using hangout of GooglePlus. Two persons participate in this conference, they are in different place, they are collaborating to edit a shared GoogleDoc as shown in the top of the screen.

4. "different time" and "different place", systems in this region support distributed asynchronous collaboration, participants do not work simultaneously. The interaction can be produced in different time between geographically distributed participants. Examples include communication systems such as electronic mails and online forums, coordination systems such as workflow [84] and shared agendas. Many Web 2.0 systems such as Wikis and blogs are classified in this category. Wikis are online editors that allow people to edit a wiki page while they are distributed in time and in space. Figure 2.4a shows an example of a Wikipedia page. This page describes the city of Nantes. In figure 2.4b, we can see the history of modifications of this page, this helps users to understand what happened to this page. Different contributors from different places around the world can edit this page at different time.

2.1 Multi-Synchronous Collaboration Model

Dourish [23] introduced the multi-synchronous collaboration model and defined it 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 collaborative systems do not fit in the CSCW matrix [78]. For instance, it is possible for two participants to work at same time, in same place but on different copies with divergence i.e. workspaces share the same objects but are not equal at the same time. Divergence between worskpaces is the fundamental difference between synchronous, asynchronous and multi-synchronous collaboration models.

Multi-synchronous collaboration model can be seen as a general collaboration model where synchronous and asynchronous collaborations are special cases of this general model [23]. The type of the specialization is defined by the period of synchronization. The period of the synchronization is the regularity with which two streams are synchronized, this determines the length of time that two streams will remain divergent. This can varying from milliseconds to periods of weeks or more. When the period is very small, the synchronization happens frequently and therefore divergence before reaching a consistent state is small. This is similar to the characteristic of "real-time" or synchronous groupware where participants work simultaneously in the some workspace and communicate their actions as they happen. When the period of divergence is measured in hours, days or weeks i.e. synchronization is less frequent in comparison with user activity, the divergence will increase, this is similar to asynchronous interaction.

(a) Nantes's page in Wikipedia

en.wikipedia.org, by Edits, with Page = Nantes

show 100 / 250 / 500 / 1000 | next 500 >>

Edits ↓	User	first edit	last edit
1 (1/0)	130.212.173.91 (anon)	2009-05-07 19:21	2009-05-07 19:21
1 (1/0)	94.172.115.29 (anon)	2010-11-02 19:37	2010-11-02 19:37
1 (1/0)	12.220.24.225 (anon)	2006-02-04 06:25	2006-02-04 06:25
1 (0/1)	Grenavitar	2007-02-10 03:56	2007-02-10 03:56
1 (1/0)	87.38.30.2 (anon)	2007-10-22 08:18	2007-10-22 08:18
1 (0/1)	CommonsDelinker (bot)	2008-09-14 00:07	2008-09-14 00:07
1 (1/0)	Rimkam	2009-12-01 00:33	2009-12-01 00:33
1 (1/0)	204.185.188.249 (anon)	2004-02-24 15:52	2004-02-24 15:52
1 (1/0)	205.188.116.137 (anon)	2006-07-04 21:26	2006-07-04 21:26
1 (1/0)	86.214.208.213 (anon)	2007-05-21 13:24	2007-05-21 13:24
1 (0/1)	TJRC	2008-02-26 20:30	2008-02-26 20:30
1 (1/0)	130.88.155.233 (anon)	2008-11-26 12:40	2008-11-26 12:40
1 (0/1)	TobeBot (bot)	2010-05-30 02:48	2010-05-30 02:48
1 (1/0)	Rama	2005-05-22 09:03	2005-05-22 09:03
1 (1/0)	213.36.12.120 (anon)	2005-08-07 00:13	2005-08-07 00:13
1 (1/0)	Scotthatton	2006-12-04 15:20	2006-12-04 15:20
1 (1/0)	132.205.55.25 (anon)	2007-08-05 19:39	2007-08-05 19:39
1 (0/1)	Cst17	2008-06-25 10:36	2008-06-25 10:36
1 (1/0)	Keraunos	2009-07-12 05:51	2009-07-12 05:51
1 (1/0)	91.84.77.91 (anon)	2011-02-09 13:39	2011-02-09 13:39
1 (1/0)	146.176.42.190 (anon)	2006-03-10 17:22	2006-03-10 17:22
1 (0/1)	Roscelese	2007-03-13 20:45	2007-03-13 20:45
1 (0/1)	Dancingwombatsrule	2008-10-16 16:32	2008-10-16 16:32
1 (1/0)	80.88.220.5 (anon)	2010-01-29 14:39	2010-01-29 14:39
1 (1/0)	82.235.73.151 (anon)	2011-07-28 23:30	2011-07-28 23:30
1 (1/0)	213.37.56.225 (anon)	2004-08-19 03:20	2004-08-19 03:20
1 (1/0)	132.205.55.148 (anon)	2007-06-05 15:23	2007-06-05 15:23
1 (0/1)	AlleborgoBot (bot)	2008-03-16 21:14	2008-03-16 21:14
1 (1/0)	82.5.202.240 (anon)	2010-08-20 18:44	2010-08-20 18:44
1 (1/0)	108.13.32.5 (anon)	2012-04-25 17:42	2012-04-25 17:42
1 (0/1)	Vrenator	2010-09-28 18:32	2010-09-28 18:32
1 (1/0)	201.83.80.213 (anon)	2012-06-04 02:30	2012-06-04 02:30
1 (1/0)	67.174.157.43 (anon)	2005-11-07 07:24	2005-11-07 07:24
1 (1/0)	90.12.201.192 (anon)	2007-01-13 09:13	2007-01-13 09:13
1 (1/0)	59.94.243.130 (anon)	2007-09-02 18:54	2007-09-02 18:54
1 (1/0)	Erwan	2002-08-04 05:13	2002-08-04 05:13

(b) A sample of Nantes editors

Figure 2.4: Asynchronous collaboration in different place with Wikipedia

All collaborative systems with replication and synchronization functionalities are multi-synchronous collaborative systems. Examples include Distributed Version Control Systems (DVCS) [2] with git [2], Mercurial [3] and Bazaar [4], synchronization tools such as Dropbox [5] or GoogleDrive, collaborative editors such as Distributed Wikis [77] or GoogleDoc.

Multi-synchronous collaboration models are related to copy-modify-merge paradigm in software engineering and optimistic-replication (or lazy replication) in distributed systems.

2.1.1 Multi-Synchronous Collaboration Model and Software Engineering

Copy-Modify-Merge paradigm [16] in software engineering is a way to enable multi-synchronous collaboration model. It is mainly used in Version Control Systems.

According to this paradigm, there is a shared repository that stores multi-version shared object and each participant has private workspace (see figure 2.5).

1. First, each developer creates a workspace and copies a configuration of

[2] http://git-scm.com/
[3] http://www.selenic.com/mercurial/
[4] http://bazaar.canonical.com/
[5] http://www.dropbox.com

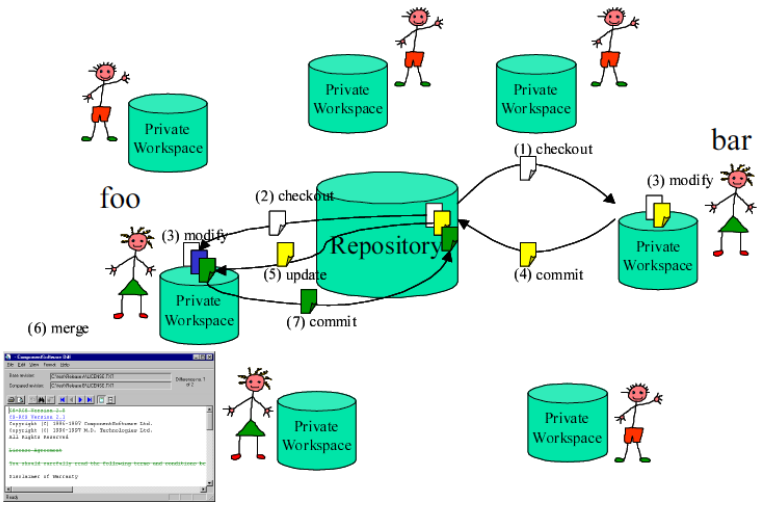


Figure 2.5: Copy-Modify-Merge paradigm (source [57])

files using a "checkout" command. A configuration corresponds to an aggregate of files where, for each file, one version is selected. This corresponds to the creation of a stream of activity in the multi-synchronous collaboration model.

2. Then, each developer can make local changes to these files. This corresponds to parallel activities in multi-synchronous collaboration model. As workspaces are isolated, divergence occurs between workspaces.

3. Next, developers publish their changes, by "committing" local changes. Commit fails if the local copy is not up-to-date with the shared repository i.e. concurrent changes have been committed to the repository. In this case, developers need to update local copy and merge manually conflicts that can occur during this merge process. It clearly corresponds to the synchronization stage of multi-synchronous collaboration model. We must note that merge is performed locally. Once up-to-date, commit will succeed.

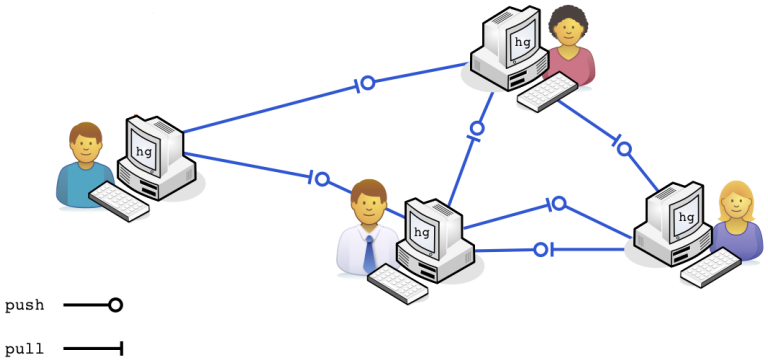


Figure 2.6: Muti-synchronous collaboration in DVCS

With the raise of Distributed Version Control models [2] such as git, Mercurial, Bazaar, copy-modify-merge evolved to a new paradigm that we will call

push-pull-clone. In this paradigm, a workspace can follow the changes published in any other workspace. This "follow your change" relations between two workspaces allow developers to create a social network oriented for multi-synchronous editing (see figure 2.6). Users of DVCS interact thanks to three main operations: *clone/push/pull*.

1. The *clone* operation allows users to create a local repository of an existing repository. This corresponds to the creation of a stream of activity in multi-synchronous collaboration model.

2. Developers can work isolated on their local repository. In this stage divergence occurs between workspaces.

3. The *push* operation allows to make public the local modifications. Unlike the commit operation in copy-modify-merge paradigm, the push operation always succeed. In pure DVCS, only the current workspace can push, so no concurrent pushes can occur.

4. Finally the *pull* operation allows to integrate remote modifications. This operation calls merge operators and generates conflicts. This corresponds to the synchronization stage of multi-synchronous collaboration model.

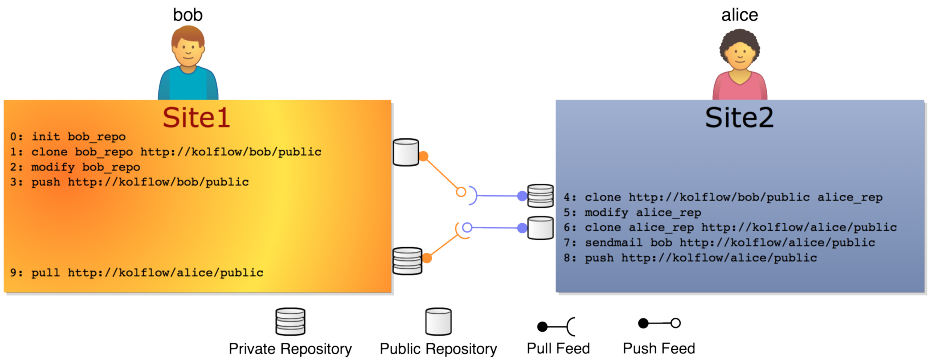


Figure 2.7: Collaboration Scenario in DVCS

Concretely, when developers use a DVCS software they can work as in the scenario depicted in figure 2.7. In this scenario, two developers *bob* and *alice* working on two different sites.

0. *bob* initializes his private repository.

1. *bob* clones his private repository into a public one so he can publish his local modifications on it.

2. *bob* modifies his private repository locally.

3. *bob* publishes his modification into his public repository.

4. *alice* wants to collaborate with *bob* on the same project. She clones *bob* public repository into her own private repository.

5. *alice* modifies her private repository locally.

6. *alice* creates a public repository by cloning her private repository to publish her modifications.

7. *alice* communicates her public repository URL to *bob* (by email for example).

8. *alice* pushes the modifications done on her private repository to her public repository.

9. *bob* pulls the modifications done on the public repository of *alice* into his private repository. This allows to maintain the two repositories synchronized and reduces the divergence.

Copy-modify-merge and push-pull-clone have important differences:

1. Copy-modify-merge is centralized and supposes one copy of reference if versioning is linear. Multiple reference copies are possible through the notion of branches. Anyway, all branches are available in the shared repository. Push-pull-clone is decentralized and multiple copies of reference are available.

2. In copy-modify-merge, all developers are known from the central repository i.e. the membership is established. In push-pull-clone, there is no global knowledge about participants i.e. a developer just known his neighbors. Membership is not required to enable multi-synchronous interactions.

For this thesis, we focus on decentralized multi-synchronous collaboration models i.e. collaboration where a central shared repository is not required.

2.1.2 Multi-Synchronous Collaboration Model and Distributed Systems

Multi-synchronous collaboration can be supported by optimistic replication models [71]. An optimistic replication model considers multiple sites hosting copies of shared objects. We can say that a site corresponds to a stream of activity. Objects can be modified anytime, anywhere by applying an update operation locally. According to the optimistic replication model:

1. Objects are modified on a site; in isolation; by generating local operations. This is the disconnection phase of the multi-synchronous collaboration.

2. Sites broadcast operations using different dissemination strategies: broadcast [29], anti-entropy [18], pairwise synchronization, or gossiping. We make the hypothesis that broadcast operations are eventually delivered.

3. Finally, sites integrate remote operations with local ones. This is the synchronization phase of the multi-synchronous collaboration; synchronization phase can generate conflicts of integration of concurrent operations.

The correctness of distributed collaborative systems belongs to weak consistency models. Some collaborative systems just ensure causal consistency [51] such as version control systems, other ensure CCI consistency such as Operational Transformation (OT) based systems [81]. The CCI consistency model is defined as:

- **Causality:** This criterion ensures that all operations ordered by a precedence relation, in the sense of the Lamport's *happened-before* relation [51], and they will be executed in same order on every site.
- **Convergence:** The system converges if all replicas are identical when the system is idle (eventual consistency).
- **Intention and Intention preservation:** The intention of an operation is the effects observed on the state when the operation was generated. The effects of executing an operation at all sites are the same as the intention of the operation.

Many algorithms that verify the CCI model have been developed and implemented [62, 79, 85]. Recently, more efficient classes of algorithms called Commutative Replicated Data Type (CRDT) have been developed [86]. Defining consistency of multi-synchronous systems is fundamental to determine what is the expected convergence state after divergence phases. Causal consistency does not force workspaces equality after synchronization but causal histories will be the same on all sites. Eventual consistency ensures workspace equality but some operation effects can be lost. Intention preservation ensures more properties on convergence state, but all intentions cannot be preserved.

Compared to software engineering models, neither copy-modify-merge nor push-pull-clone define consistency. After merge, it is not required that all workspaces are equal. Optimistic replication models can be centralized or decentralized depending on the underlying algorithms used for maintaining consistency. In next section, we describe different scenarios of multi-synchronous interactions in different domains. We illustrate how divergence occurs, how conflicts are managed and which consistency is ensured.

2.1.3 Multi-Synchronous Collaboration Scenarios

Divergence, synchronization, conflicts and consistency are important concepts of multi-synchronous collaboration model. To illustrate theses concepts, we will present three different scenarios of multi-synchronous collaboration in different contexts: collaborative edition, file synchronization and software engineering.

Multi-Synchronous Editing with GoogleDoc. Divergence can be easily observed in current well known collaborative editing systems such as GoogleDoc. In the following scenario, we show how participants can generate divergence in GoogleDoc. Imagine two persons p_1 and p_2 editing a shared GoogleDoc document. Each participant has a copy of the shared document. At the beginning their copies are identical, they have the text: "Divergence allows parallel stream of activities." The two copies are convergent, later, p_1 decides to work disconnected, while p_2 continues to edit the document online. While disconnected p_1 inserts the text "Divergence awareness is important." at the end of his copy of the shared document. At the same time p_2 inserts the text "Divergence metrics quantifies divergence" at the end of the document. p_1 and p_2 edit concurrently the shared document. During this period, p_1 and p_2 will not see the same text, divergence occurs, as shown in the figure 2.8. During

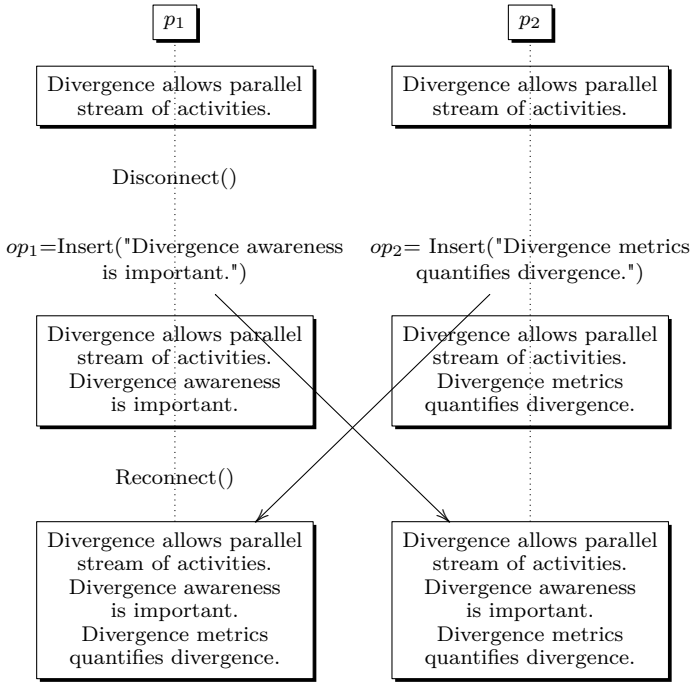


Figure 2.8: Convergence/Divergence in GoogleDoc scenario with two participants

the disconnection of p_1, his modifications will not be visible to p_2 and vice-versa. This means that the two copies do not have the same value, they are divergent. Divergence can be seen as the editing distance between the copies of p_1 and p_2.

When p_1 reconnects, the two copies will be synchronized. In GoogleDoc synchronization is done by using OT based algorithm [81, 56]. Transformation functions will find a convergent state without conflict stage solving. OT algorithm will also preserve CCI consistency. However, the convergent state might not fit users' expectations. In our scenario, modifications done by p_1 and p_2 will be integrated as shown in the figure 2.8. The two copies are convergent and the activity of the group will progress.

The same scenario of collaboration can be generalized to a group of three or more participants. Some members of the group can work connected and others can work disconnected. Imagine three persons are sharing a GoogleDoc document. Each participant has a copy of the shared document. At the beginning their copies are identical, they have the text: "Divergence allows parallel streams of activities." as in the previous scenario. p_1 decides to work disconnected while p_2 and p_3 decide to continue to work connected. During the disconnection of p_1 his copy will diverge with respect to the copies of the other group member. Whereas the copy of p_2 and p_3 still convergent. Modifications done by p_2 are immediately propagated and integrated into the copy of p_3 and vice-versa as shown in the figure 2.9. The period of synchronization is too small, divergence is negligible. The group can reach convergent state only after the connection of p_1. This scenario demonstrates how multi-synchronous collaboration model can integrate synchronous and multi-synchronous interactions smoothly.

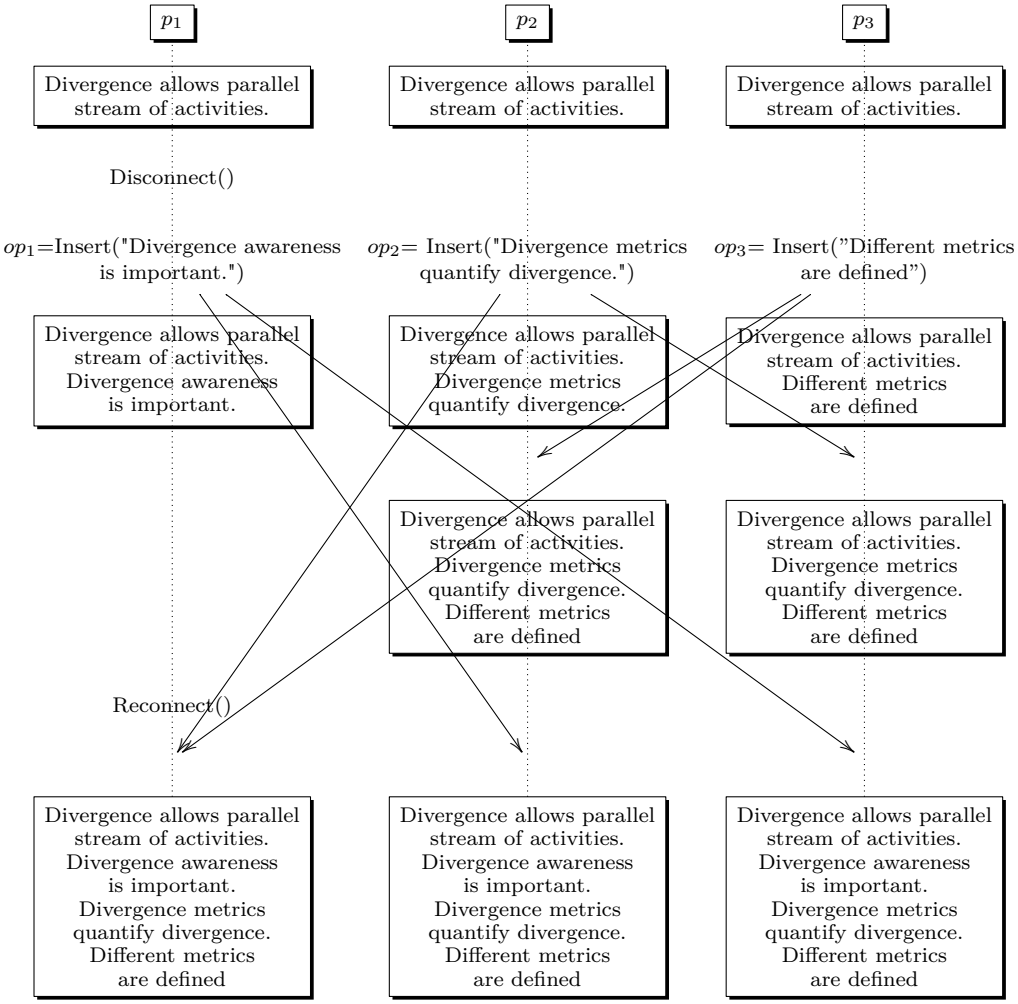


Figure 2.9: Synchronous and Multi-synchronous Collaboration in GoogleDoc

Multi-synchronous editing with Dropbox. Imagine two persons p_1 and p_2 use Dropbox to edit a shared file called "1_1background.tex" (see figure 2.10). At the beginning the file contains the line: "Rules of Acquisition." Each person has her own copy of the document.

1. At the beginning both copies are identical.

2. Next, p_1 and p_2 disconnect their workspaces and concurrent modifications are realized in each workspace.

3. Later, workspaces are reconnected and synchronization takes place. Dropbox will keep in "1_1background.tex" modifications of of p_1 and moves into "1_1background.tex (copy of p_2) in conflict 2012-07-06.tex" concurrent modification of p_2 as shown in the figure 2.11. Both workspaces are converging to this state and users can perform manual merge later.

Compared to the previous scenario, we can observe that Dropbox synchronizes files at the file system level and does not try to merge file content. It generates conflicts that can be very costly to manage. In comparison, GoogleDoc manages documents at character level and don't generate conflicts. Dropbox ensures eventual consistency i.e. all workspace will always finish to converge even if they contain many conflicts.

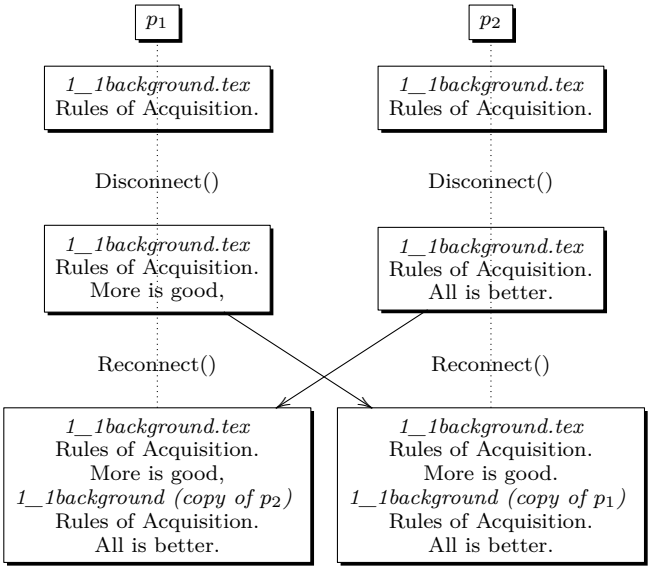


Figure 2.10: Multi-Synchronous Collaboration in Dropbox

Multi-synchronous editing with Version Control Systems. The last scenario [46] is in the domain of software engineering and Version Control Systems (VCS). Imagine three software developers collaborating on the source code of the same project using *git* or *Mercurial*. Each developer has a workspace and works on an individual copy of the source code files. Each developer repeatedly makes changes to her local copy of the files, share this changes with the team, and incorporates changes from other team members.

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. In $step_1$ of the scenario presented in figure 2.12, $developer_1$ decides to remove the method isReal() from the class Integer, for instance, in the file *Integer.java*. Concurrently, $developer_2$ modifies her copy of the file *Integer.java*, she 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(). In $step_2$, $developer_2$ and $developer_3$ receive the delete operation of $developer_1$. The integration of the delete operation with the local changes of $developer_2$ and $developer_3$ will generate conflicts. The integrated file contains block of conflicts that indicate the name and the location of conflicting operations. Consequently, $developer_2$ and $developer_3$ have to work to resolve the conflicts manually.

$developer_2$ decides to (re)insert the method isReal(), and $developer_3$ decides to remove the test method since the isReal() has been deleted. In $step_3$, $developer_3$ receives the operation of insertion from $developer_2$. A block of conflict is generated now $developer_3$ has to (re)write the test method for isReal().

Finally, there is a lot of conflict resolution and a lot of wasted work: $developer_1$ deleted the method isReal() which was reinserted by $developer_2$. His work was useless and produced side-effects for the tasks of other developers. $developer_2$ modified the method isReal() but due to its removal by $developer_1$ he needed to re-perform his initial change. $developer_3$ wrote the test for method isReal() and was obliged to remove it. Then again he had to re-write it again, he performed his work twice.

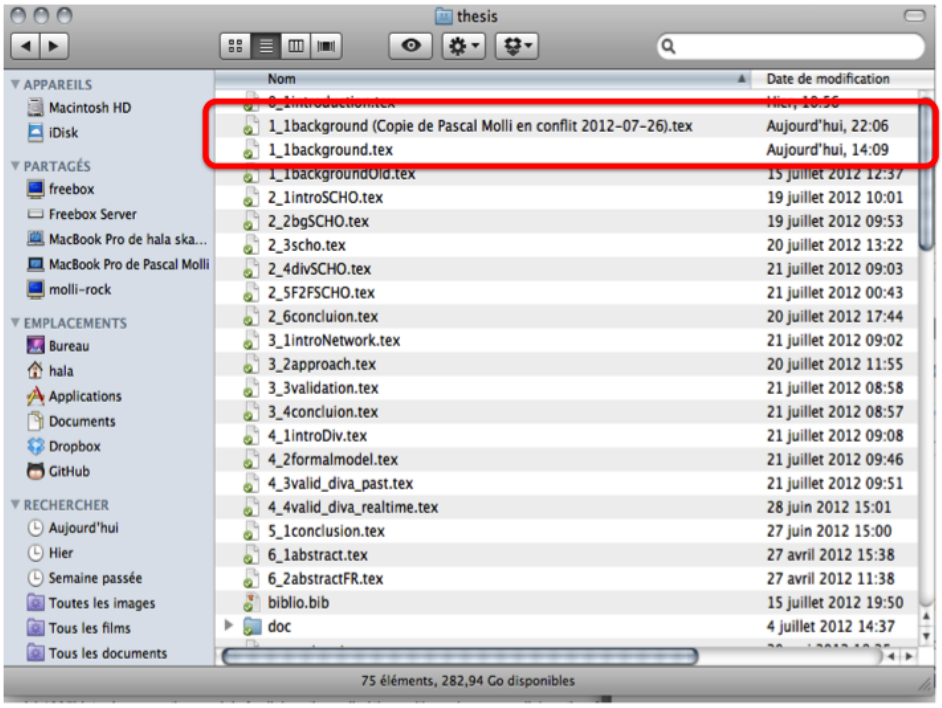


Figure 2.11: Conflict Detection in Dropbox

This scenario demonstrates issues related to blind modifications. Working without any knowledge about concurrent activities can lead to lost work and complex conflict resolution.

2.1.4 Multi-Synchronous Collaboration Model Issues

Multi-synchronous collaborative systems aim to reduce task completion time with parallelization. However, if conflicts are difficult to resolve, then the expected benefits of tasks' parallelization can be lost.

Several approaches exist to limit divergence and reduce conflicts in multi-synchronous collaboration system:

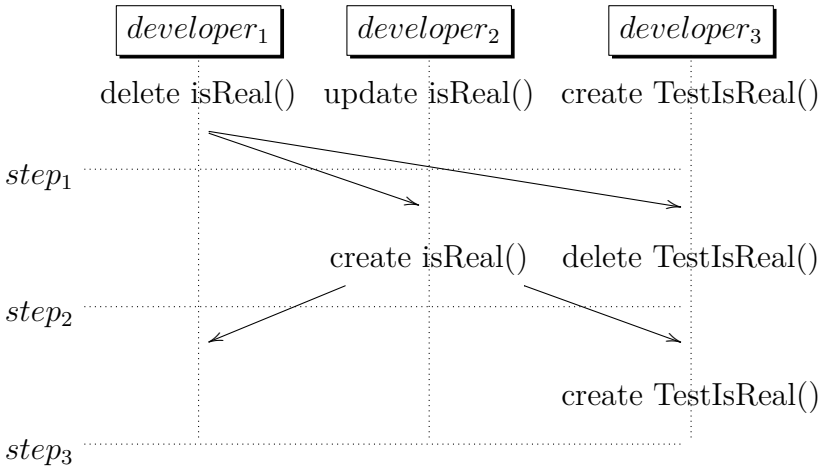


Figure 2.12: Three developers collaboration scenario

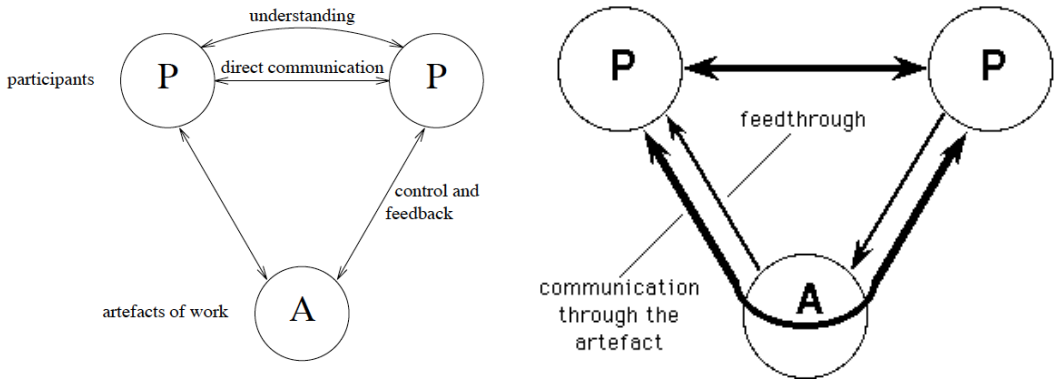
- Planning and coordination can be used to avoid conflicts [20, 28]. By good planning, one can create different parallel tasks that will modify disjoint and independent objects. But fine grained planning can be very costly, and it is not always possible to define disjoint tasks. Furthermore, in the open 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 [88]. 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 planing and commit policies are good practices, conflicts still exist as detailed in [89, 15]. Zimmermann [89] analyzed CVS repositories, and concluded that, 23% to 47% of all merges had textual conflicts. A textual conflict arises when two developers make concurrent changes to the same part of the source code. For instance, in the scenario 2.12, there is a textual conflict between the $developer_1$ and $developer_2$. Brun et al. [15] 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.

- Divergence awareness [58] can be used with planning and coordination. Awareness allows participants to establish a mutual understanding to perform joints actions. The hypothesis of divergence awareness is that if participants are aware about divergence in the group, they can avoid complex conflicts solving situation. Divergence awareness helps to prevent blind modifications that can lead to complex conflicts solving and useless work. Divergence awareness measures divergence and visualize it in order to help users answering the following questions: is there any divergence? With whom? Where? And how much?

In this thesis, we focus on divergence awareness for decentralized multi-synchronous collaborative systems.

2.2 Divergence Awareness

According to Alan Dix [22, 21] a cooperative work involves: participants and artifacts. Participants are the individuals that are working, and artifacts are the objects on which the participants work. Participants can perform direct communication between each other and they can control artifacts and get feedback from these artifacts. They can also communicate indirectly through the artifacts, as shown in figure 2.13. The communication and understanding concepts are linked to each other. It is necessary to communicate when performing an action using a groupware to manage task execution. It is essential too for each participant to be aware of others' actions, running or completed tasks, in order to coordinate her actions and tasks in accordance to the other participants of this groupware. Direct and indirect communication help participants to establish a mutual understanding. This allows participants to perform joint actions and to progress the group activity. This mutual understanding is called



(a) Communication and control (b) Feedback and feedthrough

Figure 2.13: Alan Dix's Collaboration model (source [21])

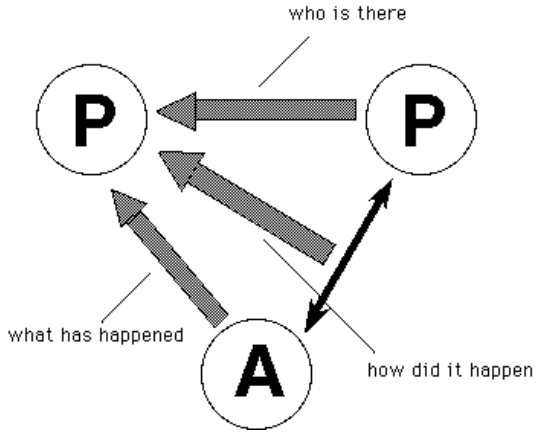


Figure 2.14: Awareness according to Dix (source [22])

awareness. According to Dix [22], awareness as shown in figure 2.14, tries to inform the participant about who else is there? What has happened to an artifact? And how did it happen? Dourish [24] proposed another definition for awareness. *Awareness is an understanding of the activities of others, which provides a context for your own activity.* Using the context allows to ensure that individual actions are compatible with the collaboration goals.

Providing awareness about individual and group activities is essential for successful collaboration. Awareness is commonly supported in CSCW systems by gathering information explicitly generated and separated from the shared object or passively collected and presented in the shared collaborative space as the object of collaboration.

2.2.1 Awareness in CSCW

Awareness plays a number of key roles:

1. First, high-level awareness of the character of others' actions allows participants to structure their activities and avoid duplication of work.

2. Second, lower-level awareness of the content of others' actions allows fine-grained shared working and synergistic group behavior which needs to be supported by collaborative applications.

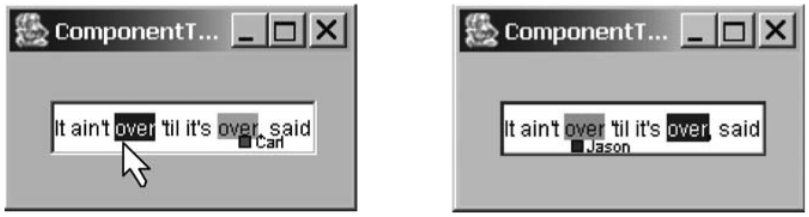


Figure 2.15: GTextField from the MAUI toolkit

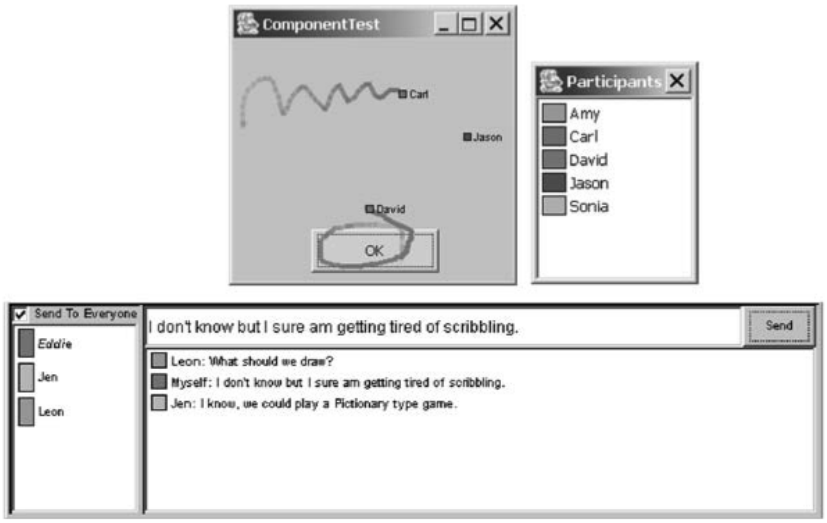


Figure 2.16: Telepointers, participant list, and chat tool from the MAUI toolkit

Awareness can be delivered to participants using special widgets like the ones proposed by Hill et al. [44] in the MAUI toolkit. MAUI shows people's activities as they manipulate the application interface. This is what Dix [22, 21] called feedthrough-feedback to the single user that also helps others understand the activity. For example, watching another person navigate through the items in a menu gives valuable clues about what they intend to do next. In figure 2.15, we have an example of a textfield for group where the text is shared, but selections in that text are multi-user, and are shown as colored transparent overlays. This means that the text has a single state for all members of the group, and modification by any person changes the state of the text for everyone. However, each person can select different parts of the text. MAUI provides also telepointers, participant lists, and a chat tool. These components show a variety of awareness information including who is in the session, where they are working, and how active they are. Participants' names can be added to the pointer representation. In figure 2.16, telepointers gives the users two sources of information about others' activities and intentions in the interface: the feedthrough information provided by the widgets, and the embodiment information provided by the telepointer. In addition, MAUI provides participant list, a simple component that shows the names and colors of all connected users. It also provides communication support. Finally, the chat tool allows messages to be directed to specific participants or broadcast to all.

Different kinds of awareness are supported by CSCW systems such as workspace awareness [24], change awareness [82], and Divergence awareness [58].

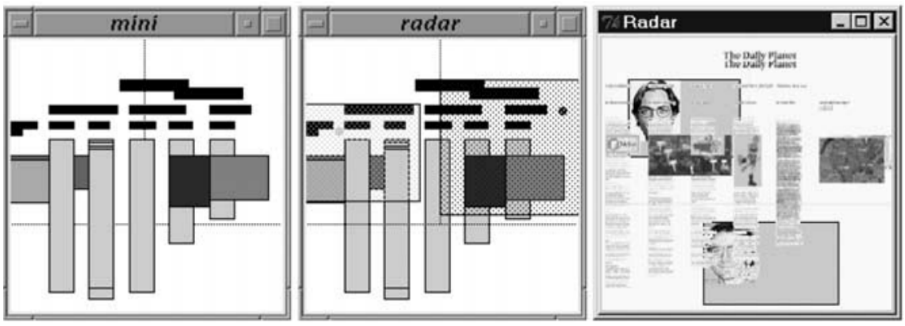
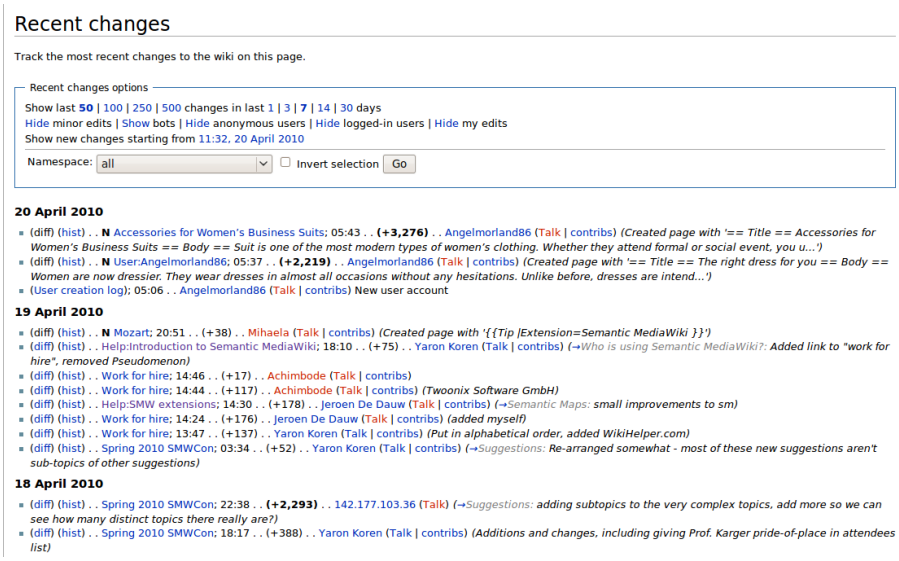


Figure 2.17: Workspace awareness as Radar View


Recent changes

Track the most recent changes to the wiki on this page.

Recent changes options

Show last 50 | 100 | 250 | 500 changes in last 1 | 3 | 7 | 14 | 30 days
Hide minor edits | Show bots | Hide anonymous users | Hide logged-in users | Hide my edits
Show new changes starting from 11:32, 20 April 2010

Namespace: all Invert selection Go

20 April 2010

- (diff) (hist) . . N Accessories for Women's Business Suits; 05:43 . . (+3,276) . . Angelmorland86 (Talk | contribs) *(Created page with '== Title == Accessories for Women's Business Suits == Body == Suit is one of the most modern types of women's clothing. Whether they attend formal or social event, you u...')*
- (diff) (hist) . . N User:Angelmorland86; 05:37 . . (+2,219) . . Angelmorland86 (Talk | contribs) *(Created page with '== Title == The right dress for you == Body == Women are now dressier. They wear dresses in almost all occasions without any hesitations. Unlike before, dresses are intend...')*
- (User creation log); 05:06 . . Angelmorland86 (Talk | contribs) New user account

19 April 2010

- (diff) (hist) . . N Mozart; 20:51 . . (+38) . . Mihaela (Talk | contribs) *(Created page with '{{Tip |Extension=Semantic MediaWiki }}')*
- (diff) (hist) . . Help:Introduction to Semantic MediaWiki; 18:10 . . (+75) . . Yaron Koren (Talk | contribs) *(→Who is using Semantic MediaWiki?: Added link to "work for hire", removed Pseudomenon)*
- (diff) (hist) . . Work for hire; 14:46 . . (+17) . . Achimbode (Talk | contribs)
- (diff) (hist) . . Work for hire; 14:44 . . (+117) . . Achimbode (Talk | contribs) *(Twoonix Software GmbH)*
- (diff) (hist) . . Help:SMW extensions; 14:30 . . (+178) . . Jeroen De Dauw (Talk | contribs) *(→Semantic Maps: small improvements to sm)*
- (diff) (hist) . . Work for hire; 14:24 . . (+176) . . Jeroen De Dauw (Talk | contribs) *(added myself)*
- (diff) (hist) . . Work for hire; 13:47 . . (+137) . . Yaron Koren (Talk | contribs) *(Put in alphabetical order, added WikiHelper.com)*
- (diff) (hist) . . Spring 2010 SMWCon; 03:34 . . (+52) . . Yaron Koren (Talk | contribs) *(→Suggestions: Re-arranged somewhat - most of these new suggestions aren't sub-topics of other suggestions)*

18 April 2010

- (diff) (hist) . . Spring 2010 SMWCon; 22:38 . . (+2,293) . . 142.177.103.36 (Talk) *(→Suggestions: adding subtopics to the very complex topics, add more so we can see how many distinct topics there really are?)*
- (diff) (hist) . . Spring 2010 SMWCon; 18:17 . . (+388) . . Yaron Koren (Talk | contribs) *(Additions and changes, including giving Prof. Karger pride-of-place in attendees list)*

Figure 2.18: Change awareness

Workspace awareness [40]: is awareness about people and how they interact with shared workspace. It was originally designed for real-time groupware, it delivers information about "who, what and where". Who is currently present in the shared workspace? What are they currently doing? Where are they currently working or looking? Awareness is delivered to participants through specified widgets such as radar views, telepointers and multi-user scrollbars [32]. An example of workspace awareness delivered through radar views is shown in the figure 2.17. Radar views, as defined by [40], are secondary windows used with a detailed view of the shared workspace; they show miniatures of the artifacts in a shared workspace, and can also be used to show awareness information about the participants in the session. Figure 2.17 presents three versions of the GroupKit radar views, the first one shows the object movement only; the second one adds location information by showing each person's main view as a shaded rectangle; the last one adds photographs for participant identification. As we can see, these radar views allow to answer questions about: Who? What? Where?

Change awareness [82]: is related to workspace awareness for past interactions. It provides information about past events in the workspace. It allows to answer the question: is anything different since the last time I looked at the work? The online collaborative editors; such as wikis; provide change aware-

ness, this awareness is presented as a wiki page. For each wiki page, there is a corresponding page that maintains the history of the modifications of the page. Figure 2.18 shows an example of the recent changes to a wiki page in Wikipedia, this page shows change awareness. The wiki system monitors user edition and records the date of the modifications, the modification and the responsible of this modification in a special page.

Divergence awareness [58]: is designed for multi-synchronous collaborative systems. If the system is idle; at every site, there is no local operations to publish nor remote ones to integrate at every site; divergence is null in the system. Otherwise, divergence can be quantified using ad-hoc metrics based on the number of operations produced, disseminated or integrated in the system. Divergence awareness allows to answer the questions [58]: is there any divergence? With whom I diverge? Where is the divergence is located? And how much?.

We distinguish two kinds of divergence awareness: *divergence awareness for the past* and *real-time divergence awareness*:

1. *Divergence awareness for the past* checks objects states and consistency after the synchronization and integration phase. Divergence metrics are computed for past interactions. This awareness can be seen as a kind of change awareness for multi-synchronous collaboration.

2. *Real-time divergence awareness* informs participants about modifications in progress on other sites and consequently participants are fully aware about conflicts risks. This awareness prevents complex conflicts and preserves the natural benefits of multi-synchronous collaboration. Divergence metrics are computed for real-time interactions, they alert the participants of potential conflicts. Only this awareness prevents blind modifications [46]. This awareness can bee seen as a kind of workspace awareness for multi-synchronous collaboration. In workspace awareness divergence is nearly invisible, however, in multi-synchronous collaboration, divergence is explicit, participants work isolated on their copies and synchronize from time to time.

Computing divergence metrics after synchronizations relies on data that can be accessed locally. While computing divergence metrics before synchronization is more challenging and requires a distributed computation with all related distributed problems such as scalability, availability and privacy preservation.

In the following section, we detail some existing divergence awareness systems.

2.2.2 Divergence Awareness Systems

Divergence occurs when there are more than one copy of a shared document and participants can modify their copies in parallel. Existing divergence awareness systems are characterized by their basic multi-synchronous collaboration model, divergence metrics, when, and how they are computed.

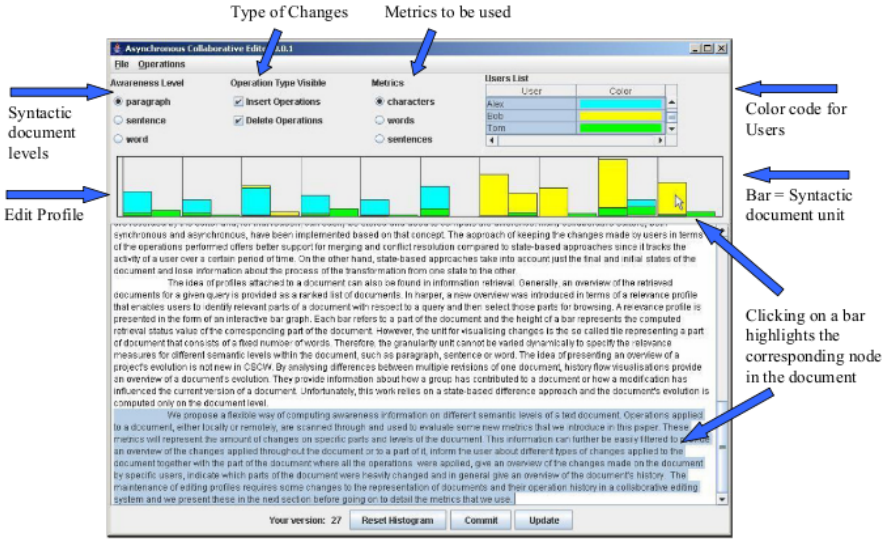


Figure 2.19: Edit profile (source [63])

Edit Profile [63] is a multi-synchronous collaborative editor extended with awareness about activity of users. Edit profile makes users aware of "hot areas" and also who is or has been active in various parts of the document as shown in the figure 2.19. The different contributions of users are quantified and distributed at different levels: document, paragraph, sentence, word, and character. The participant has the possibility to choose at which level she needs to be aware of the changes on a document. So the metrics are calculated based on the participant choice of details. The participant also has the possibility to choose which type of operations she is interested in. For example; insert or delete operations. The system assigns a different color for each participant to distinguish his/her contribution from the others. It is possible to observe who contributed where, and how much. Edit Profile metrics can be customized according to the document structure to deliver more readable awareness.

Even, if Edit Profile quantifies and displays contributions of different users, it does not quantify divergence i.e. it does not quantify concurrent operations. Edit Profile is more related to change awareness than divergence awareness.

Concurrency awareness [3] is designed for a peer-to-peer network of synchronized wikis. A peer-to-peer network of wikis follows the optimistic replication model described in section 2.1.2 and consequently, behaves as multi-synchronous collaborative editor. In such network, a wiki page can be edited anywhere, anytime on any wiki server. In case of concurrent changes, automatic merge is performed when operations are received on each site. Consequently, a wiki page can be the result of an automatic merge with no human reviews. Concurrency awareness aims to make users aware about wiki pages that have been merged automatically and to locate the effects of the merge on the page. Concurrency awareness relies on plausible clocks [83] to detect concurrency *a posteriori*. This helps users to quickly find where automatic merges have been performed. Figure 2.20 shows an example of concurrency awareness in Wooki [85], a peer-to-peer wiki system. In this example, we see the wiki server Wooki2, in this server, a server-produced wiki page is requested by a user, an awareness visualization mechanism delivers awareness information to

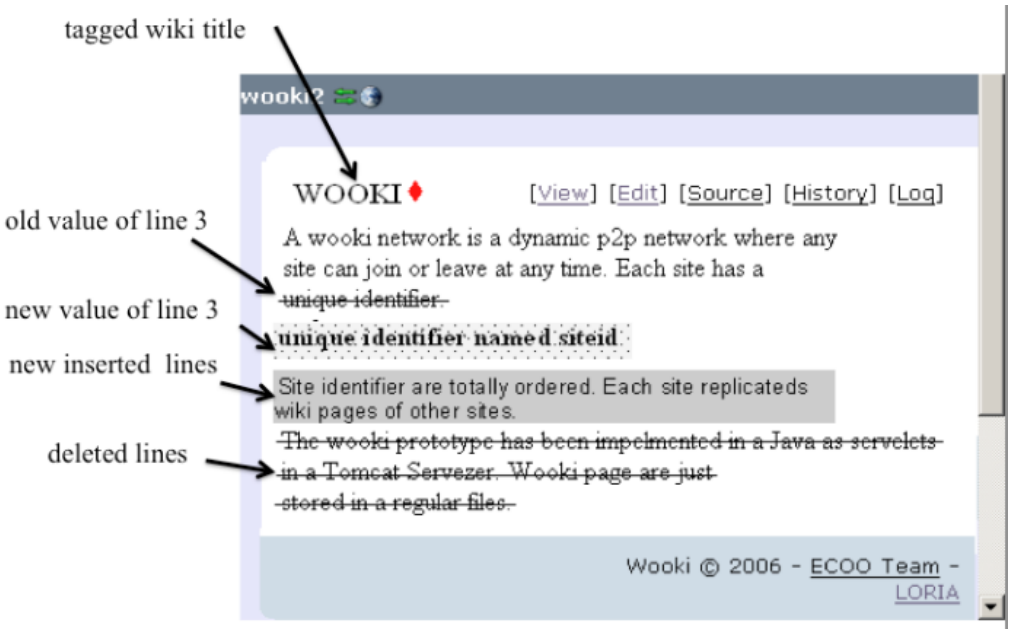


Figure 2.20: Concurrency Awareness (source [3])

the user by highlighting the effects of the concurrent part of the history in the page it returns. In the above example, the red square indicates that an automatic merge holds on this page i.e. this page has been merged and no human user reviewed it. The line 3 of the initial page has been updated. So line 3 appears two times: the first occurrence corresponds to the old value and appears overridden with a thin line, while the second occurrence corresponding to the new value appears with a colored background. Two other lines have been inserted, these lines appear also with a colored background. The last line is deleted so it is overridden with a thin line. The other lines: lines 1 and 2 appear normally since they are not impacted by the concurrent history at this stage. Concurrency awareness primary objective is not to avoid blind modifications but to alert people where concurrency occurred. Concurrency awareness metrics are calculated on already integrated operations, therefore, it is a divergence awareness of the past.

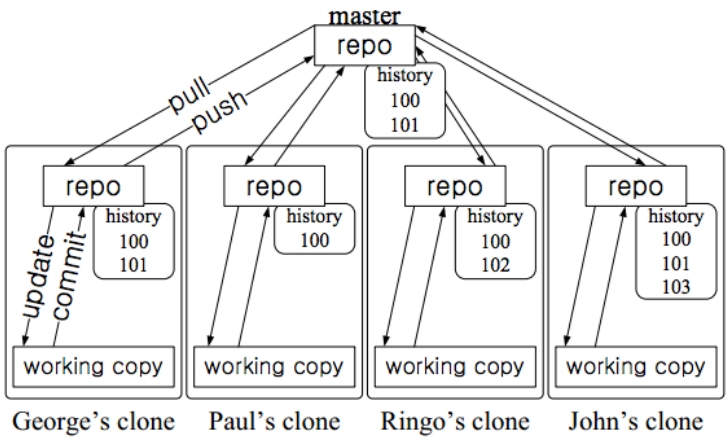


Figure 2.21: Crystal setting for DVCS (source [15])

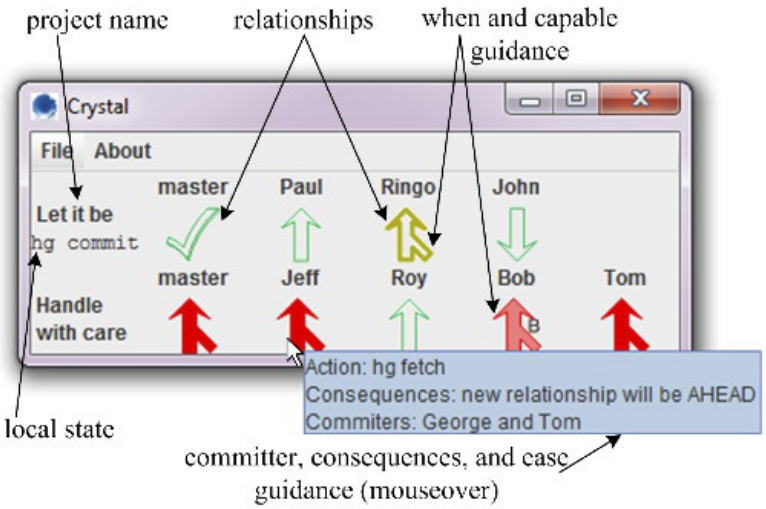


Figure 2.22: Crystal widget for divergence awareness (source [15])

Crystal [15] is a set of metrics built on top of a version control system. It aims to give advice about pending conflicts while remaining largely unobtrusive. It can be used in the context of both centralized and distributed version control systems. For distributed version control systems, it considers a single *master* repository and local developers' repositories as shown in figure 2.21. In this example, there is a single *master* repository and four developers: George, Paul, Ringo, and John. At the beginning, each developer makes a local repository by cloning the master. Each local repository contains a complete and independent history of the master repository at the time it was cloned. In addition, each local repository has a working copy in which code is edited. Different commands enable developers to publish their local changes to the master, or to consume other published changes. Crystal provides a developer with information on his development state and the relationships between his repository and collaborators' repositories. It shows the developer if he's in advance over the other's i.e. her changes are integrated in the master or if he is behind i.e. changes have been made to the shared project and he did not consume them yet. It also alerts the developer of the potential conflicts in case he consumes the remote operations (see figure 2.22). Figure 2.22 shows the Crystal view of George. George is involved in two development projects: "Let it be" and "Handle with care". The first project has four collaborators: George, Paul, Ringo and John; the second one has five collaborators: George, Jeff, Roy, Bob and Tom. The figure shows George's local state and his relationships with the master repository and the other collaborators, as well as guidance based on that information. The color of each relationship icon represents the type of relationship. For instance, the green arrow informs the developer that his changes can be published without conflict to the master repository. The red merge symbol indicates to the developer that publishing his changes will generate conflicts. Crystal needs access to that developer's repository and the locations of the all other collaborators' repositories. Actually Crystal creates a dedicated repository for awareness computation and it integrates all the collaborators' modifications into it. Although Crystal results can be applied in distributed version control system, divergence awareness in

Crystal is computed with respect to a reference copy i.e. the master copy.

Crystal provides divergence awareness: it helps to locate where divergence is located and with who. However, divergence is not really quantified, it is impossible to measure the total amount of divergence in the system. Crystal track divergence by checking the state of all workspaces according to a master copy. If multiple reference copy are required, it will be assimilated to different projects and can produce confusions for users.

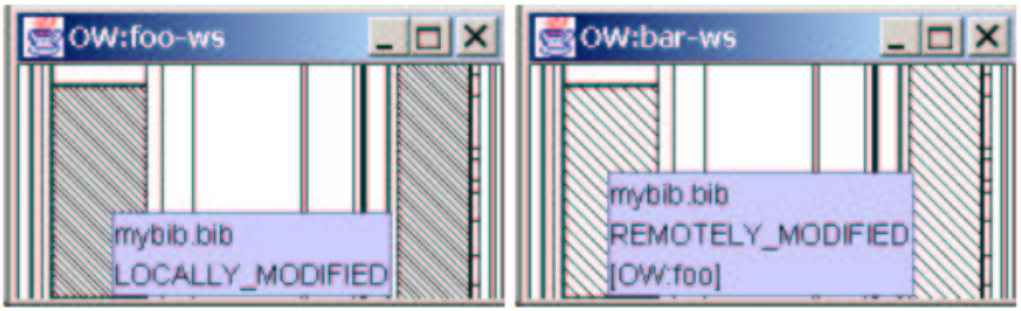


Figure 2.23: State Treemap (source [57])

State Treemap [57] is a divergence awareness widget for the copy-modify-merge paradigm (see section 2.1.1). It informs participants about states of shared documents in their workspace according to a shared repository. Developers start by copying files from the shared repository, they make local changes to these files and then they commit their changes to the shared repository if they are up-to-date. They have to synchronize local workspace with last version of the shared repository otherwise. State Treemap define different states for each document in the local workspace:

- *Locally Modified* enables the participant to know that her own copy was modified where the others are not.
- *Remotely Modified* makes the participant aware of the changes that occur in the remote workspaces.
- *Need Update* means that a new version of the document is available.
- *Potential Conflict* means that more than one participant are updating the same document.

When a document is modified by a participant, it will be marked as LocallyModified in her own workspace, where 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 (see figure 2.23). For instance, if the whole treemap is white, this means that there is no divergence in the system. If some parts are colored, then users know who changed the file i.e. owner "foo" in figure 2.23. 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 divergence in the system. In

State Treemap, different users do not see the same treemap (see figure 2.23). The quantification of divergence as the number of rectangle of different colors will be different. The quantity of divergence in the global system depends on the workspace of the user.

Although State Treemap clearly belongs to divergence awareness; it helps users to locate divergence and with whom. Quantification of divergence is not the primary objective of State Treemap. There is no clear definition that help to compute the total amount of divergence in the system. Uncommitted changes are handled through the "remotely modified" state. However, if local workspace is not up-to-date, the "need update" state will mask the "remotely modified" state. As Crystal, State Treemap relies on a reference copy and gives no guidelines on how to evaluate divergence in a decentralized multi-synchronous system.

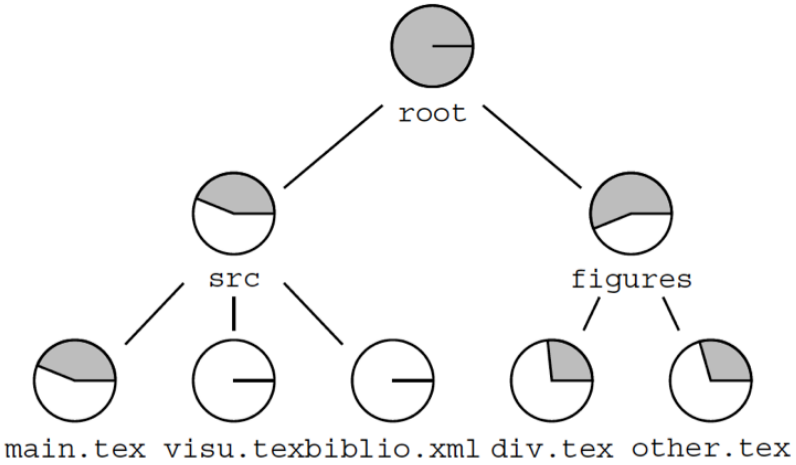
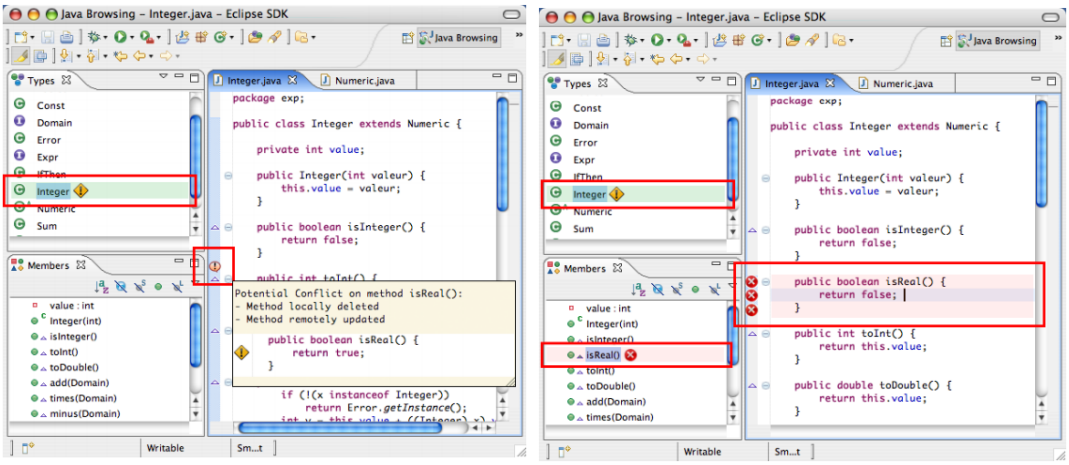


Figure 2.24: Operational Transformation divergence awareness (source [58])

OT Divergence Awareness [58] aims to quantify divergence in multi-synchronous collaborative editor based on operational transformation [81] . For convenience, we will call it OT divergence awareness. This approach follows the optimistic replication approach with one exception: local operations are sent on demand but representatives of local operations are sent immediately in real-time. When these fake operations are received, an operational transformation algorithm that simulates the integration 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 shown in figure 2.24. Users can know the total amount of divergence in the system and where it is located.

Unlike Crystal or State Treemap, OT divergence awareness is not defined according to a reference copy. However, one important issue with operational transformation approach is that it should guarantee that all sites will compute the same conflict objects and next will give the same size on each site. It requires to develop quite complex transformation functions and prove convergence properties on them. In addition, a system can be divergent even with no conflicts. In this case, a conflict based metric will not capture it.



(a) $developer_1$ interface after integrating ghost operation of $developer_2$

(b) $developer_2$ interface after integrating ghost operation of $developer_1$

Figure 2.25: Ghost operations Divergence awareness (source [46])

Ghost operations [46] authors introduced ghost operations to deliver real-time divergence awareness following a version control system approach in Eclipse IDE. Ghost operations represent real unpublished operations, some parameters of operations can be blurred according to user's preferences to better preserve privacy. To illustrate this divergence awareness, we will use the scenario given in figure 2.12, where three software developers collaborate on the same source code of a project. $developer_1$ and $developer_2$ decide to send ghost operations describing their activity while working in isolation and $developer_3$ decides to apply a strong privacy policy and does not send any ghost operations on his activity. $developer_1$ sends the full content of his modifications as ghost operations. $developer_2$ decides to apply the privacy policy that hides the content of his changes but shares their location. Figure 2.25a shows the divergence awareness at the site of $developer_1$ after receiving the ghost operation of $developer_2$. The ghost operation of $developer_2$ contains information about the target file, and the position of the modification. Therefore, the class Integer is tagged as concurrently modified and the position of the modified line is computed by using the line number indicated by the ghost operation. Figure 2.25b shows the divergence awareness at the site of $developer_2$ after receiving the ghost operation of $developer_1$. $developer_2$ is notified that the method isReal() is deleted. The lines composing this method are annotated. Knowing that the method isReal() had been deleted by the $developer_1$ will prevent $developer_2$ to do useless work.

Ghost operations aims to prevent blind modifications while preserving privacy. It can locate divergence and with whom. Ghost operations have been designed for quantifying divergence. As OT divergence awareness, Ghost operations are not dependent on a copy of reference.

Palantir [73] is a divergence awareness tool for version control system that provides software developers with insight into others' workspaces. It is designed for configuration management systems such as CVS. It enhances awareness by continuously sharing information regarding operations performed by all developers. The tool specifically informs a developer about who changes

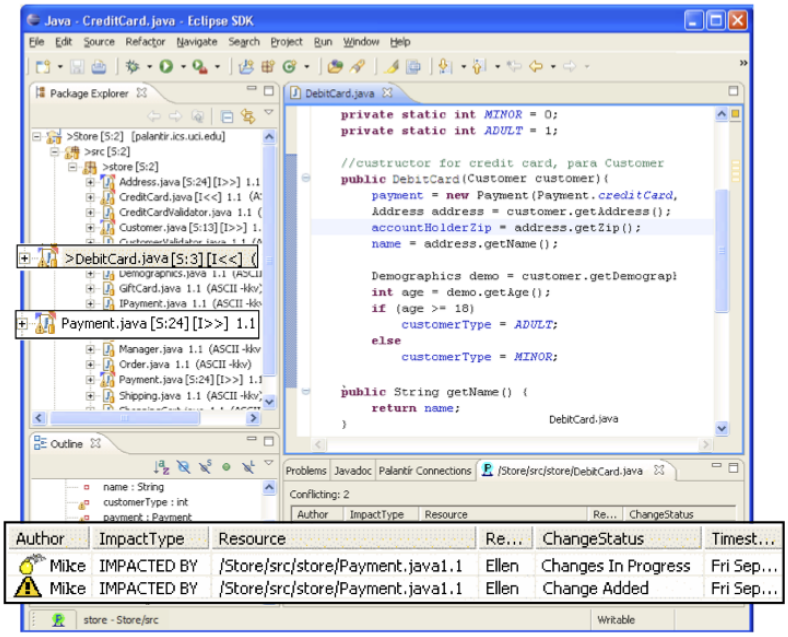


Figure 2.26: Palantir divergence awareness (source [73])

Event	Meaning
Populated	Artifact has been placed in a workspace
Synchronized	Artifact has been synchronized with repository
ChangesInProgress	Artifact has changed in the workspace
ChangesReverted	Artifact has been returned to it's original state
ChangesCommitted	New version of artifact has been stored in the repository
SeverityChanged	Amount of changes to an artifact

Table 2.1: Palantir divergence awareness states

which artifacts (see figure 2.26), focusing on the concept of conflicts. Conflicts can be direct i.e. concurrent changes on the same artifact or indirect through dependencies between files. Furthermore, Palantir provides a measure of the severity of those changes and graphically displays the information in a configurable manner. Severity of changes are based on ratio of lines changed, added or deleted according to total number of lines. Palantir is built on top of an event system, where some events are representing operations performed by users in their respective workspaces.

```
changesInProgress= {
 artifactID= [cvsroot/Store/src/
  store/Payment.java:1.2::00-0D-56-F6-
  10-7C:Ellen:003987::cvsroot/ .../store
 diffXML= [XML...]
 max_sev= 500
 actual_sev= 15 }
```

The above event represents an unpublished operation, with severity set at 15, artifactID contains causality information representing version number in CVS (1.2). Different projections according to various meta-data can be performed locally such as conflict interpretation and impact analysis. Palantir does not really define a distance, it tries to estimate the size of direct or indirect conflict as in operational transformation divergence awareness [58]. Table 2.1 summarizes the awareness states defined in Palantir proposal.

Palantir can locate divergence and with whom. It is quite similar to State Treemap and Crystal approaches with its dependency to copy-modify-merge paradigm. It also tries to quantify divergence through measuring the conflicts as in OT divergence awareness. However, divergence should be quantified without conflicts and Palantir will not capture this case.

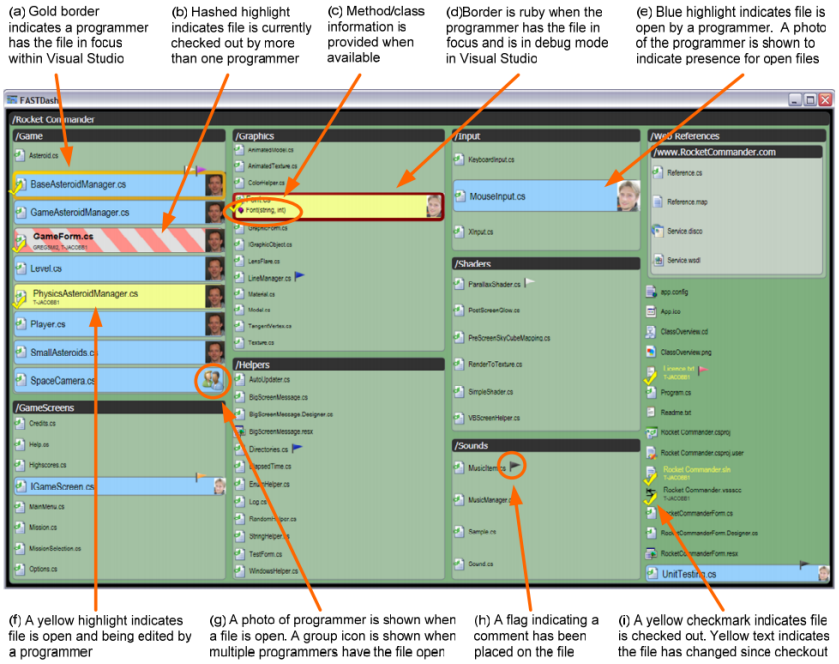


Figure 2.27: FASTDash visualization tool (source [12])

FASTDash [12] is a visualization tool that seeks to improve team activity awareness using spatial representation of the shared code base that highlights team members' current activity. Figure 2.27 visualizes a project runtime environment. It shows the active file and source code of the project and the two active programmers. As we can see, FASTDash provides many awareness information, for instance, it can answer the questions: Who is opening the file? Who is editing the file? Did the file has been modified after it has been checked? FASTDash is designed for project teams of 3-8 collaborating programmers and can enable obtainment of contextual awareness information such as which code files are changing, who is changing them and how they are being used.

FASTDash raises possible divergence between the different copies of the source code file, it does not quantify this divergence and in addition the awareness mechanism does not scale for large-scale decentralized systems.

2.3 Synthesis

Multi-synchronous collaborative systems are able to support complex interactions. They can reduce tasks completion time if conflicts or wasted work are managed. Planning and coordination can be completed with divergence awareness systems to avoid blind modification and complex conflict solving. Many divergence awareness for multi-synchronous systems have been proposed and implemented. They rely on different models, some relies on optimistic replication models and other rely on paradigms such as copy-modify-merge or push-pull-clone. Some require a copy of reference, others do not. This illustrates the lack of a clear abstract, formal, multi-synchronous model where it is possible to reason about divergence. The next issue concerns the lack of clear definition of divergence itself. If many systems are able to locate divergence and give some informations about who is involved, they poorly answer the "how much?" question. Is it the size of conflicts in the system? The sum of all editing distance between all workspaces and a reference copy? What if there is many reference copy as in decentralized multi-synchronous collaborative systems?

We address the issues related to define a model for writing divergence metrics in Chapter 3. We define a common ontology for multi-synchronous collaborative system, called SCHO. Then we rewrite all existing divergence awareness metrics as semantic queries over this ontology. Divergence awareness metrics are not embedded in the system, they are defined declaratively as queries on this ontology. Consequently, divergence metrics are no more dependent of application models.

Divergence awareness computation requires to know *a priori* unpublished operations on all the sites participating in the collaborative activities. This means that we need to run distributed queries on all those sites. The discovery of all sites involved in collaboration is an issue in decentralized multi-synchronous collaboration models. This membership issue is addressed in Chapter 4, where we propose the extension of SCHO ontology to take in consideration membership by linking the different "rdf files" generated on each site and adding new inference rules. Each site publishes its network information to its collaborators (or neighbors). Then we can run a distributed query using the Link Traversal Query Based Execution. The advantage of this approach that it does not require a beforehand knowledge of the sites that it will seek for getting the data to evaluate the query result, which is compatible with the multi-synchronous collaboration model. Unfortunately, after experimenting with the Link Traversal Query Based Execution approach we found that it does not scale for a large distributed collaborative system. We need to find an alternative to compute divergence awareness efficiently. In order to achieve this, firstly, we need to formally define what is divergence awareness for a group of collaborators. Secondly, we need an efficient algorithm to compute the group divergence awareness metrics. Divergence awareness has not been formalized before, all existing proposals are ad-hoc implementations for certain collaboration software. Existing divergence metrics rely on different interpretation of editing distance between copies in the system, counting numbers of unpublished operations, concurrent operations, concurrent conflicting operations, etc. They do not formally define what they calculate which make them incomparable and we can not infer the properties that they guarantee.

Computing divergence awareness in real-time is very challenging, it requires to query remote states of all participants and next aggregates information to compute a global metric. This raises severe problems of performances, reliability and dynamism of participants. The definition of a formal model and efficient computation algorithms are addressed in Chapter 5. In this chapter, we propose a formal model GroupDiv to define formally divergence awareness. Then we propose efficient algorithms to calculate the group divergence awareness.

3

SCHO: Shared Causal History Ontology

Contents

3.1 Introduction

Many previous work have addressed the measurement and the visualization of divergence in multi-synchronous collaborative systems. Divergence awareness is provided in different systems with ad-hoc visualizations, such as State Treemap [57], Palantir [72], Edit Profile [63], and Wooki [3].

Existing systems define their own divergence metrics without a common formal definition. Metrics are coupled with the application and cannot be

htb]

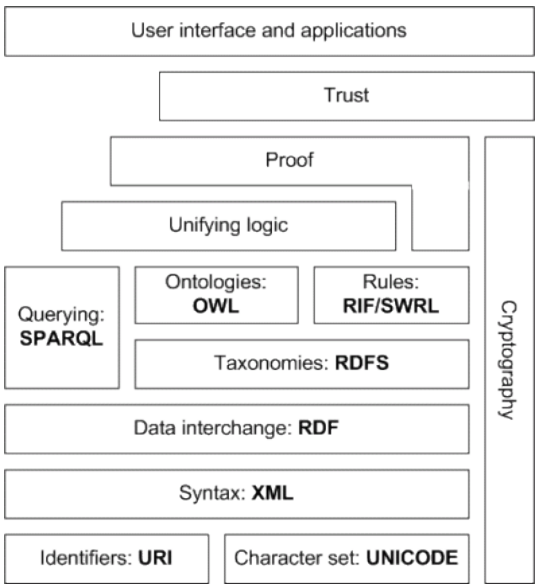


Figure 3.1: The Semantic Web stack

used outside their original scope. There is no previous work that tried to build a unified formal model for divergence awareness. A unified model for computing divergence opens the opportunity to build a middleware for distributed collaborative systems.

Multi-synchronous collaborative systems rely on an optimistic replication model [67]. The optimistic replication model is based on history sharing which is application independent. We propose to conceptualize and formalize the sharing of causal history in multi-synchronous collaborative systems. This allows to compute divergence metrics in a declarative way independently of the application.

In this work, we use semantic web technologies to define an ontology for constructing and sharing the causal history in a distributed collaborative system. Then, we define the existing divergence metrics in a declarative way as semantic queries over this ontology. Finally, we validate our work using real data extracted from software engineering development projects.

3.2 Semantic Web and Ontologies

The Semantic Web [10, 75] is an effort to extend the current web so that the presented information can be better processed by machines and software agents which, subsequently, will allow improvements of current functionalities provided on the web. The underlying idea of the Semantic Web is to represent knowledge on the web in a machine-processable form which explicitly captures the meaning of the presented information. To achieve that, the existing information sources such as web pages, images or videos are extended with explicit statements which specify the meaning of the presented information. Such statements need to be expressed in some formal language suitable for knowledge representation.

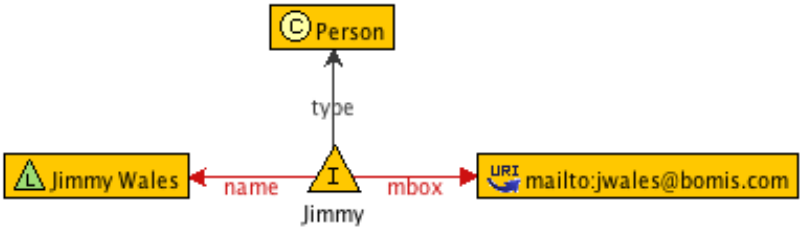


Figure 3.2: RDF graph representation

As part of the Semantic Web effort several languages for annotating information on the web have been developed. Among these languages, the Resource Description Framework (RDF) [53], the RDF Schema (RDFS) [13] and the Web Ontology Language (OWL) [35] have a prominent role. All these three languages are part of a stack of W3C recommendations, shown in figure 3.1. Building on shared common principles, such as identifying resources and objects by Unified Resource Identifiers (URIs), or supporting an XML serialization. Such principles allows an easy integration with other existing W3C web standards.

The Resource Description Framework describes resources by introducing statements called triples, where each statement has a form of a simple triple consisting of a subject, a predicate and an object, as shown in listing 3.1. By using triples it is possible to describe resources in terms of properties and property values. A set of triples can be represented as a graph with nodes and arcs representing resources and their properties with values, as shown in figure 3.2.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix example: <http://url.com/firends#> .

example:Jimmy  rdf:type example:Person .
example:Jimmy  example:name "Jimmy␣Wales" .
example:Jimmy  example:mbox <mailto:jwales@bomis.com> .
```

Listing 3.1: RDF n3 representation

```
<?xml version="1.0"?>
<rdf:RDF xmlns:example="http://url.com/firends#"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
 <example:Person rdf:about="http://url.com/firends#Jimmy">
  <example:name>Jimmy Wales</example:name>
  <example:mbox rdf:resource="mailto:jwales@bomis.com" />
 </example:Person>
</rdf:RDF>
```

Listing 3.2: RDF/XML representation

RDF defines both a graph-based data model based on subject, predicate, and object triples, and the RDF/XML format through which an RDF graph of one or more triples can be serialized as an XML document, as shown in listing 3.2. The subject of any RDF triple must be a URI or a 'blank node', the predicate must be a URI, and the object can be either a URI, a Literal or a blank node.

Publishing data in RDF conveys a number of benefits: data is machine-readable, easily integrated for querying or other forms of processing, and easily linked across disparate sources. Traditional data formats such as Comma Separated Values (CSV), XML and even HTML can all be described as machine-readable, as data can be represented in these formats and parsed reliably by software applications. However, data represented in RDF is machine-readable in a different way. Not only it is machine-readable at a syntactic level (i.e. it can be parsed reliably) but also at a semantic level, in that the meaning of RDF data is made explicit. The meaning of data described in RDF is indicated by the use of classes and properties (relations) taken from shared ontologies available on the Web and identified by a URI.

The RDF Schema extends RDF with mechanisms for specification of RDF vocabularies. RDF Schema defines classes and properties that may be used to describe classes, properties and other resources. Specifically, classes and properties might be defined in terms of a subclass relation between classes, a sub-property relation between properties, and domain and range constraints for properties. RDFS presents a very basic language for terminology specification. An example is shown in listing 3.3.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix example: <http://myurl.com/firends#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

example:person rdf:type      rdfs:Class .
example:man    rdf:type      rdfs:Class .
example:man    rdfs:subClassOf example:person .
example:woman  rdf:type      rdfs:Class .
example:woman  rdfs:subClassOf example:person .
example:mother rdf:type      rdfs:Property .
example:mother rdfs:domain   example:person .
example:mother rdfs:range    example:woman .
example:father rdf:type      rdfs:Property .
example:father rdfs:domain   example:person .
example:father rdfs:range    example:man .
```

Listing 3.3: RDFS example

RDF triplets are usually stored in special databases called triplestores, such as (Apache Jena [1], Sesame [2], and Virtuoso [3]). To retrieve and manipulate data stored in a triplestore a W3C recommendation was adopted in 2008 called SPARQL [4]. SPARQL allows for a query to consist of triple patterns, conjunctions, disjunctions, and optional patterns. For example the query in listing 3.4, searches for all the subjects of type Person, and return their names and emails.

[1] jena.apache.org
[2] www.openrdf.org
[3] virtuoso.openlinksw.com
[4] www.w3.org/TR/rdf-sparql-query/

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?email
WHERE {
  ?person rdf:type foaf:Person.
  ?person foaf:name ?name.
  ?person foaf:mbox ?email.
}
```

Listing 3.4: SPARQL query example

The Web Ontology Language (OWL) provides much stronger mechanisms for terminology definitions along with a formal semantics. In addition to classes and properties definitions supported in RDFS, OWL introduces constructs for definition of complex class descriptions based on logical operations such as intersection, union and negation. OWL also supports additional properties characteristics such as cardinality restrictions, value constraints, and so on. OWL defines three increasingly-expressive sub-languages: OWL-Lite, OWL-DL, and OWL-Full. OWL-Lite and OWL-DL impose constraints on the language constructs which allows to guarantee a computational completeness and decidability and also allows the underlying language semantics to be directly based on Description Logics [8]. On the other hand, while OWL-Full is the most expressive sub-language, reasoners are not guaranteed to be complete for OWL-Full.

The majority of Semantic Web languages have their roots in research of ontologies. An ontology can be defined as a formal specification of conceptualization [36] or, more precisely, an ontology is a formal explicit specification of terms in the domain of interest and relations among them [61]. Another definition of ontology is proposed by Guarino et al. [37] as: *a logical theory which gives an explicit, partial account of a conceptualization.* They insist on the fact that an ontology can not be considered as representation of the reality, in the sense where this reality might be perceived differently depending on the point of view, the culture, etc. All the definitions of ontology introduce the notion conceptualization. Guarino et al. [37] define a conceptualization as: *an intensional semantic structure which encodes the implicit rules constraining the structure of a piece of reality.* In another definition a conceptualization [38] is considered as an abstract and a simplified view of the world that we want to represent to achieve a certain goal. An ontology is not a knowledge base [43], it defines the vocabulary and the formal specification that permits the construction of a knowledge base.

Ontologies serve for knowledge representation purposes where terms are typically called concepts or classes, while relations are usually referred to as roles or properties. Concepts with relations between them constitute the domain terminology (often referred to as an intensional knowledge), while instances of introduced concepts describe specific individuals in the domain of interest (often referred to as an extensional knowledge). As stated in [61], ontologies allow to:

- share common understanding among people or software agents,
- reuse domain knowledge,
- make domain assumptions explicit,

- separate domain knowledge from the operational knowledge, and
- reason about concepts.

There are many languages for defining ontologies with the above mentioned Semantic Web languages presenting the most recent ones. For example, the OWL language has evolved from the DARPA Agent Markup Language (DAML [5]) and the Ontology Interference Layer (OIL). The Knowledge Interchange Format (KIF [6]) is another knowledge representation language which was primarily developed for the interchange of knowledge among disparate computer systems. From the formal point of view, the formal semantics of these ontology languages usually relies on some logic formalism such as First-Order Logic, Frame Logic [50], or Description Logic [36].

3.3 SCHO: Shared Causal History Ontology

Divergence occurs when there are more than one copy of a shared document. Optimistic replication model [71] considers (N) sites sharing copies of shared document. A document is modified by executing an operation on it. Any operation has the following life cycle:

1. Generated on one site and executed locally immediately,
2. broadcasted to the other sites,
3. received by other sites and re-executed.

The causality property is essential in a collaborative system to avoid users' confusion [81]. Causality ensures that all operations are ordered by a precedence relation in the sense of the Lamport's happened-before relation [51]. Therefore they will be executed in same order on every site. Broadcasting operations is not fully determined in the general optimistic replication framework. But it is assumed that all operations should be eventually delivered to all sites. The causality and broadcasting are application independent.

We observed that divergence metrics on a document can be computed relying on the state of its operations according to the operation life cycle in the optimistic replication model. Our approach is to define an ontology to formalize concepts and relations that allow to build and to share causal histories. This ontology allows the formal definition of existing divergence metrics and to calculate them as semantic queries.

The broadcast is represented using the general approach of publish/subscribe that can be used by any distributed collaborative system [67]. The model is not application dependent. Consequently, we have to determine the underlying concepts required to exchange patches i.e. set of operations. The publish/subscribe model works as follows:

- When a document is modified on a site, patches are generated. A patch is a set of operations related to one document.

[5] http://www.daml.org/
[6] http://logic.stanford.edu/kif/dpans.html

- Several patches can be combined in one changeset that can be published into one or several channels called PushFeeds.

- An authorized site can create a PullFeed corresponding to an existing PushFeed and pull changesets. Then the patches contained in the changesets can be re-executed locally. If needed, the integration process merges this modification with concurrent ones, generated either locally or received from a remote site.

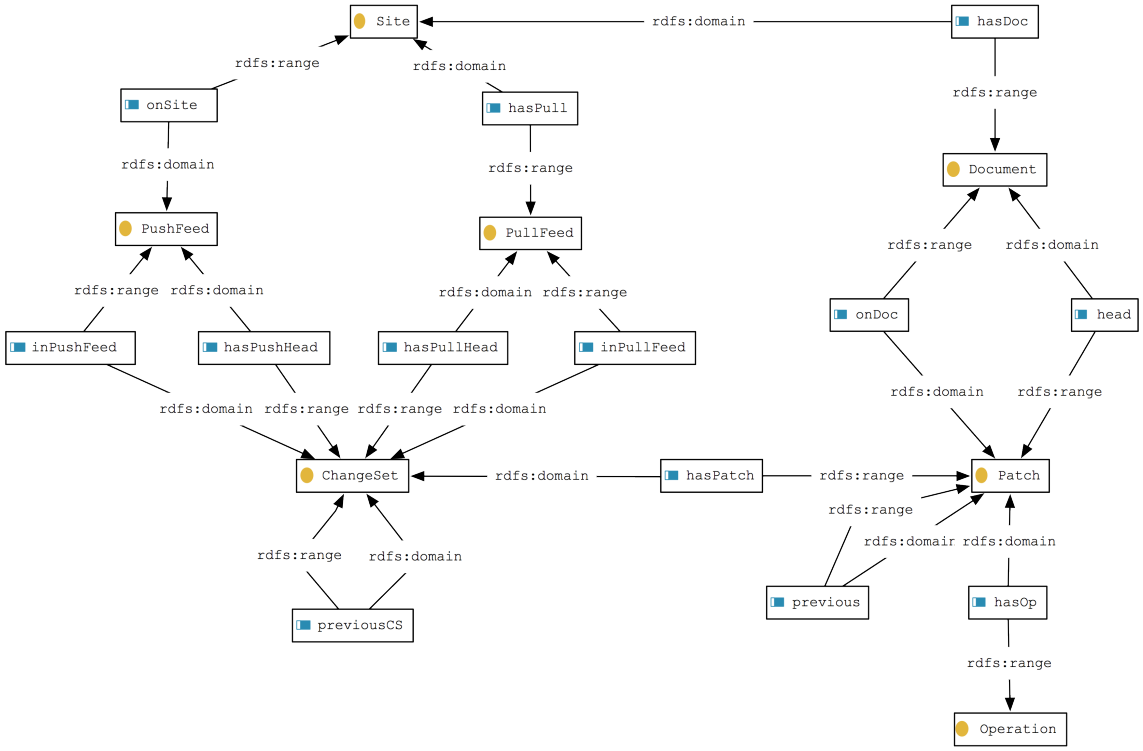


Figure 3.3: Shared Causal History Ontology

3.3.1 Unified Shared Causal History Model

The Shared Causal History Ontology (SCHO[7]) [5] shown in Figure 3.3 represents all the concepts of *SCHO*: changesets, patches, push and pull feeds. This ontology enables the *SCHO* users to query the current state of the document and its complete history using semantic queries. *SCHO* ontology is populated through users' interactions with the system using five basic operations: *createPatch*, *createPush*, *push*, *createPull* and *pull*. These operations are inspired by the Push-Pull-Clone model used in distributed version control systems such as git, Mercurial and Bazaar [2]. These operations create instances of the SCHO ontology. The details and the algorithms of each operation are presented in the following section.

Each site can perform five operations:

- createPatch: Generates operations,

[7] http://kolflow.univ-nantes.fr/mediawiki/images/scho+.owl

- createPush: Creates a feed in the Push-Pull-Clone model,
- push: Publishes local operations on the feed,
- createPull: Subscribes to a remote feed,
- pull: Consumes remote operations.

These five operations enable the building and sharing of causal history.

The OWL file for the Shared Causal History Ontology (SCHO) is provided in Appendix A.

The *SCHO* ontology defines basic concepts common to Push-Pull-Clone model such as ChangeSet, Patch, Previous, Operation, etc. It also defines more precise concepts such as PullFeed and PushFeed. These concepts allow the distinction between published/unpublished operations and consumed/unconsumed operations which is essential for computing divergence awareness. The advantage of using *SCHO* ontology is that we have a unified minimal ontology for representing and managing the shared causal history of any Push-Pull-Clone system. Divergence awareness metrics are now calculated using declarative queries independent from the application and not hard-coded into it. This will make it easier to develop universal tools and plug-ins for any Push-Pull-Clone based system that adopts this ontology for managing its causal history or what is commonly called *log*.

- *Site*: This concept has the following attributes and properties:
  - siteID: This attribute contains the identifier of the site.
  - hasPull and hasDoc : The range of these properties are respectively a *PullFeed* and a *Document*. A site has several PullFeeds and several Documents.
- *Document*: This concept has the following attributes and properties:
  - docID: This attribute contains the identifier of the document.
  - head: This property points to the last *Patch* applied to the document.
- *Operation*: This concept represents a change in a document. An operation has the following attributes:
  - operationID: This attribute contains the unique identifier of the operation.
- *Patch*: A set of operations. A patch is calculated during the save of document. A patch has the following properties:
  - patchID: A unique identifier of the patch.
  - onDoc: The range of this property is the *Document* where the patch was applied.
  - hasOp: This property points to the *Operations* generated when the document was saved.

  - previous: This property points to the precedent executed *Patch* on the local site.

- *ChangeSet*: This concepts is defined as a set of patches. It is important to support transactional changes. It allows to group patches generated on multiple documents. Therefore, it is possible to push modifications on multiple documents. It has the following attributes and properties:

  - changSetID: A unique identifier of a *ChangeSet*.
  - hasPatch: This property points to the *Patches* generated since the last push.
  - previousCS: This property points to the precedent *ChangeSet*.
  - inPushFeed: The range of this property is a *PushFeed*. This property indicates the *PushFeed* that publishes the *ChangeSet*.
  - inPullFeed: The range of this property is a *PullFeed*. This property indicates the *PullFeed* that pulls a *ChangeSet*.

- *PushFeed*: This concept is used to publish changes made on a site. It has the following properties:

  - pushID: A unique identifier of the *PushFeed*.
  - onSite: The range of this property is a *Site*.
  - hasPushHead: This property points to the last published *ChangeSet*.

- *PullFeed*: This concept is used to receive the changes made on a remote Site. It has the following attributes and properties:

  - pullID: A unique identifier of the *PullFeed*.
  - hasPullHead: This property points to the last pulled *ChangeSet*.

The *SCHO* ontology is managed by the five operations mentioned earlier. The algorithms for these operations are presented in the following section. These algorithms ensure that each site implements a causal reception [67]. Consequently, the proposed framework ensures causality.

3.3.2 Unified Shared Causal History Algorithms

The *createPatch* operation is called when a document is modified. It calls a *diff* function that computes the operations related to a particular type of document. The *diff* function is an application dependent function.

```
createPatch (int docID, int modDocID) {
       Patch pid = new Patch(concat(site.siteID,
              site.logicalClock++));
       foreach op ∈ diff(docID, modDocID) do {
              Operation opid = new Operation
               (concat(site.siteID, site.logicalClock++));
              opid.content=op;
              hasOp(pid, opid);
       }
       previous(pid, doc.head);
       head(doc, pid);
       onDoc(pid, doc);
}
```

Listing 3.5: createPatch operation

The communication between sites is made through feeds. The *createPush* operation creates a PushFeed. A PushFeed is used to publish the changes.

```
 createPush (String pushName) {
       PushFeed PF=new PushFeed(pushName);
       hasPush(site, pushName);
       Push(pushName);
}
```

Listing 3.6: createPush operation

The *push* operation creates a ChangeSet corresponding to the documents and adds it to the PushFeed.

```
Push (String pfName) {
       ChangeSet csid= new ChangeSet(concat(site.siteID,
              site.logicalClock++));
       inPushFeed(csid, pusName);
       published = set(Patch x : Changeset(y)
              && inPushFeed(y,pusName) && hasPatch(y,x));
       patches = set(Patch x : Document(p) && onDoc(p,x));
       foreach patch ∈ {patches - published } do
               hasPatch(csid,patch);
       endfor
       previousChangeSet(csid, pushName.hasPushHead);
       setPushHead(pusName, csid);
}
```

Listing 3.7: Push operation

PullFeeds are created to consume changes from PushFeeds on remote sites into the local site. A PullFeed has a corresponding PushFeed. In the sense that it is impossible to pull unpublished data. The *createPull* operation perform this task.

```
createPull (int pullID)
{
       PullFeed PF= new PullFeed(pullID);
       hasPull(site, PF);
       Pull(PF);
}
```

Listing 3.8: createPull operation

The *pull* operation fetches the published ChangeSets that have not been pulled yet. It adds these ChangeSets to the PullFeed and integrate them into the documents on the pulled site.

```
pull (PullFeed pullName) {
       ChangeSet cs = get(pullName.headPullFeed);
       while (cs != null) do {
              CS = set(ChangeSet x : inPushFeed(x,pullName));
              if (cs ∉ CS) {
                     inPullFeed(cs, pullName);
                     integrate(cs);
              }
              cs = cs.previousChangeSet;
       }
}
```

Listing 3.9: Pull operation

The "`integrate`" function in the *pull* algorithm implements a Commutative Replicated Data Type (CRDT) algorithm [87, 65, 76]. In the following section, we detail the queries that allow each user to compute divergence awareness metrics.

3.4 Divergence Awareness in SCHO

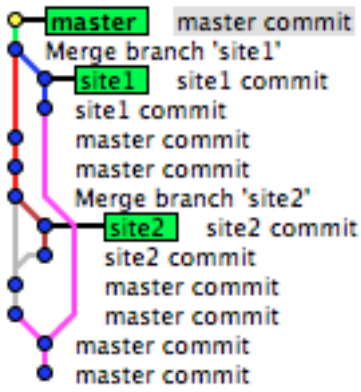


Figure 3.4: A sample project *git* history

This section details the formal definition of existing divergence awareness metrics based on SCHO ontology. We give a formal interpretation of: State Treemap, Palantir and Concurrency awareness metrics, described in the previous chapter in section 2.2.2. Other existing divergence awareness metrics can be formalized in the same way.

Divergence awareness metrics are calculated on a site for a given document. We use the following notations: *LS* to denote a local site on which divergence metrics are calculated. *RS*: to denote a remote site.

We define the following formulas:

Definition 1 (onSite(P,D,S)) *This means that a patch P belongs to a doc-*

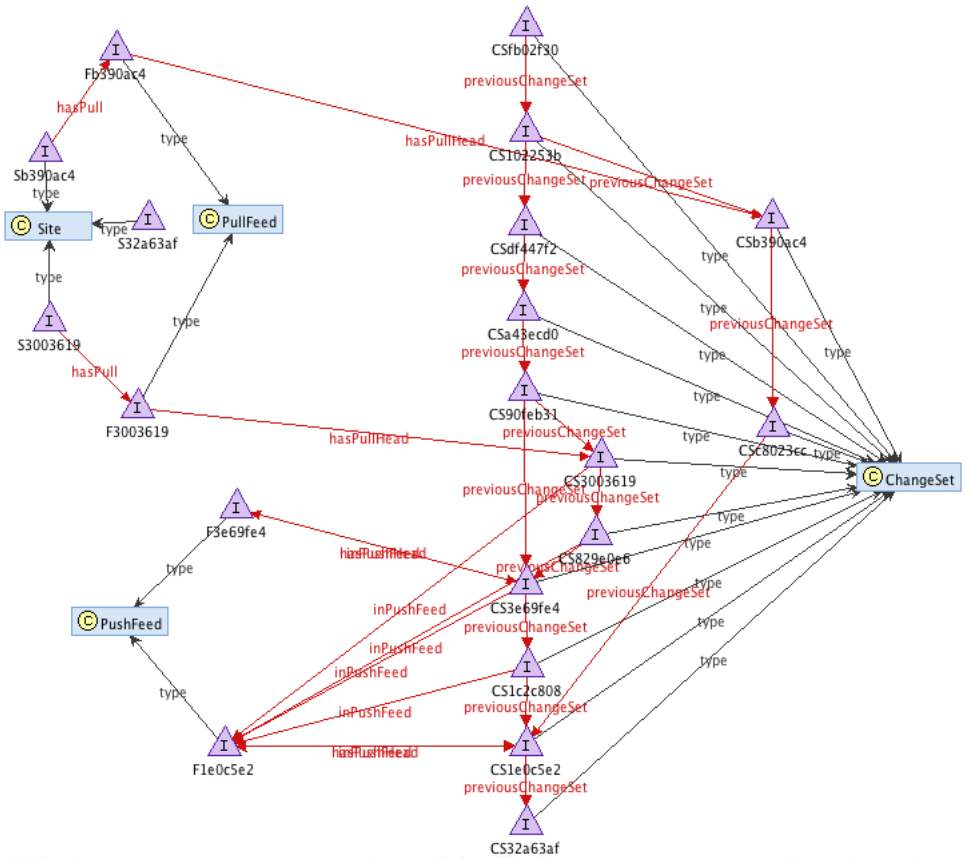


Figure 3.5: A sample project equivalent RDF graph

ument D was generated on a site S.

$$onSite(P, D, S) \equiv \exists P \exists D \exists S : \\ Patch(P) \wedge Document(D) \wedge Site(S) \wedge \\ onDoc(P, D) \wedge hasDoc(S, D)$$

Definition 2 (inPushFeed(P,S)) *This means that a patch P is published by the site S.*

$$inPushFeed(P, S) \equiv \exists P \exists PF \exists S : \\ Patch(P) \wedge PushFeed(PF) \wedge Site(S) \wedge \\ hasPush(S, PF) \wedge inPushFeed(P, PF)$$

Definition 3 (inPullFeed(P,S)) *This means that a patch P is consumed by the site S.*

$$inPullFeed(P, S) \equiv \exists P \exists PF \exists S : \\ Patch(P) \wedge PullFeed(PF) \wedge Site(S) \wedge \\ hasPull(S, PF) \wedge inPullFeed(P, PF)$$

3.4.1 State Treemap Divergence Awareness Using SCHO

To calculate the State Treemap metrics using the SCHO model, we made the following interpretations and defined the corresponding formulas.

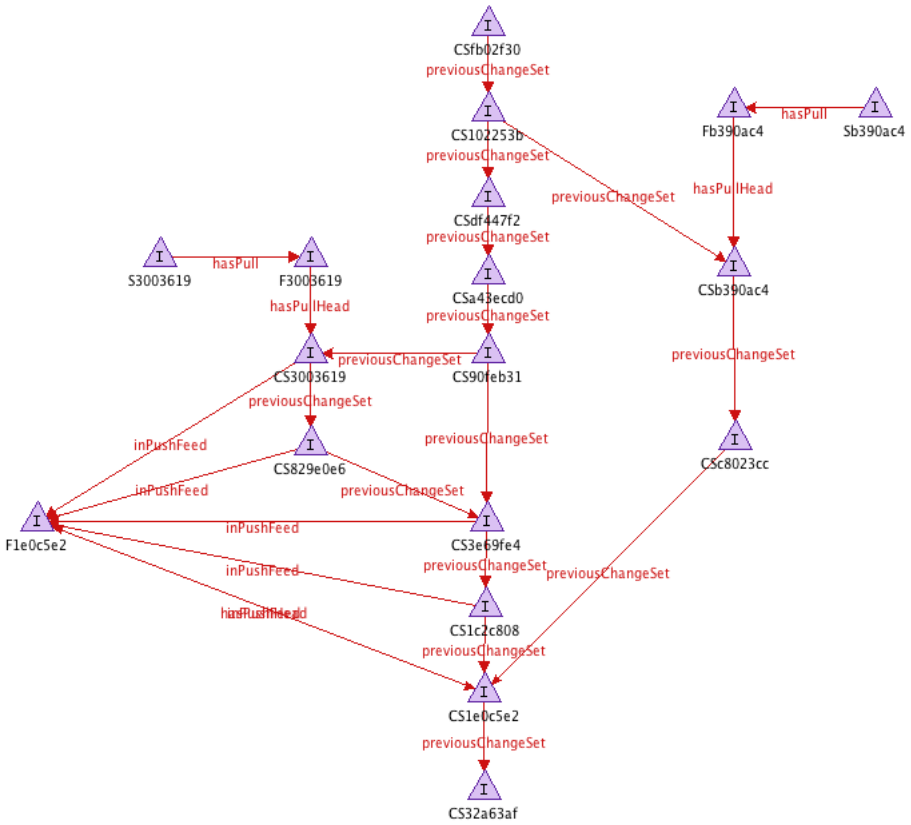


Figure 3.6: A sample project equivalent RDF graph without the concepts

Definition 4 (Locally Modified (LM)) *There are new patches in a local site which are not published in its PushFeeds.*

$$\begin{aligned} LM(D, LS) \equiv & \exists P \exists D \exists LS : \\ & Patch(P) \wedge Document(D) \wedge Site(LS) \wedge \\ & onSite(P, D, LS) \wedge \neg inPushFeed(P, LS) \end{aligned}$$

Definition 5 (Remotely Modified (RM)) *There are new patches in remote sites which are not in the PullFeeds of the current site.*

$$\begin{aligned} RM(D, LS) \equiv & \exists P \exists D \exists LS \exists RS : \\ & Patch(P) \wedge Document(D) \wedge Site(LS) \wedge Site(RS) \wedge \\ & (LS \neq RS) \wedge onSite(P, D, RS) \wedge \neg inPullFeed(P, LS) \end{aligned}$$

Definition 6 (Potential Conflict (PC)) *It is the state where the document is Locally Modified and Remotely Modified. This is the intersection of the two previous states.*

$$\begin{aligned} PC(D, LS) \equiv & \exists D \exists LS \exists RS : \\ & Document(D) \wedge Site(LS) \wedge Site(RS) \wedge \\ & (LS \neq RS) \wedge LM(D, LS) \wedge RM(D, LS) \end{aligned}$$

Definition 7 (Need Update (LNU)) *There are patches in remote sites'*

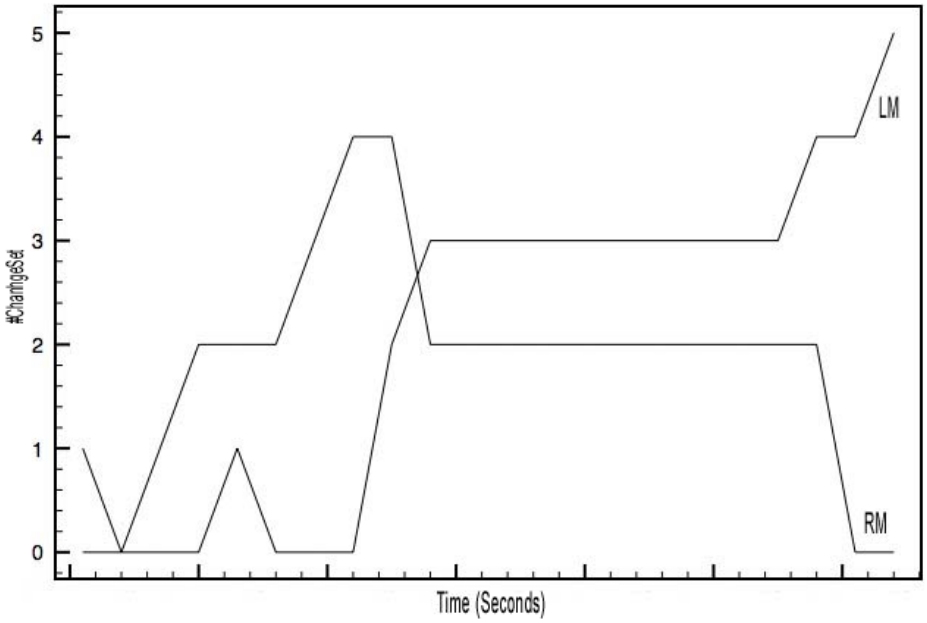


Figure 3.7: Divergence awareness for sample project

PushFeed that were not pulled locally.

$$LNU(D, LS) \equiv \exists P \exists D \exists LS \exists RS : Patch(P) \wedge Document(D) \wedge Site(LS) \wedge Site(RS) \wedge (LS \neq RS) \wedge inPushFeed(P, RS) \wedge \neg inPullFeed(P, LS)$$

Definition 8 (Will Conflict (WC))

$$WC(D, LS) \equiv \exists D \exists LS : Document(D) \wedge Site(LS) \wedge LNU(D, LS) \wedge LM(D, LS)$$

Definition 9 (Locally Up To Date (UTD))

$$UTD \equiv \forall D \forall S : Document(D) \wedge Site(S) \wedge \neg LM(D, S) \wedge \neg RM(D, S)$$

3.4.2 Palantir Divergence Awareness Using SCHO

We define Palantir's divergence awareness metrics using the SCHO ontology.

Definition 10 (Populated (Pop)) *A document has been created on a site.*

$$Pop \equiv \exists D \exists LS \forall RS : Document(D) \wedge Site(LS) \wedge Site(RS) \wedge (LS \neq RS) \wedge hasDoc(LS, D) \wedge \neg hasDoc(RS, D)$$

Definition 11 (Change in progress (CP)) *This state is similar to the Locally-Modified or Remotely-Modified states already mentioned in State Treemap.*

$$CP \equiv \exists D \exists S : Document(D) \wedge Site(S) \wedge (LM(D, S) \vee RM(D, S))$$

ChangeSet No.	State Treemap	Palantir
1	Locally Modified	Populated
2	Up to Date	Changes Committed
3	Remotely Modified	Change in Progress
4	Remotely Modified	Change in Progress
5	Potential Conflict	Change in Progress
6	Remotely Modified	Changes Committed
7	Remotely Modified	Change in Progress
8	Remotely Modified	Change in Progress
9	Potential Conflict	Change in Progress
10	Potential Conflict	Change in Progress
11	Potential Conflict	Change in Progress
12	Locally Modified	Synchronized
13	Locally Modified	Change in Progress

Table 3.1: Divergence awareness results for the sample project

Project name	DVCS	#CS	#Users	#Triples	Time (sec)
Gollum	git	613	37	2851	12
MongoDB	git	13636	91	68186	158
AllTray	Bazaar	389	3	2168	5
Anewt	Bazaar	1980	13	9433	44
hgview	Mercurial	595	15	3257	12
murky	Mercurial	198	17	1111	5

Table 3.2: Execution time and general statistics

Definition 12 (Change Reverted) *The document has returned to its original state.*

$$ChangeReverted \equiv \exists UndoOperation(LS)$$

Definition 13 (Severity Changed) *The number of patches has been done on a document.*

$$\begin{aligned} SeverityChanged &= |P_1| + |P_2| where \\ P_1 &= \{\forall p : Patch(p) \wedge Document(D) \wedge Site(LS) \\ &\quad \wedge onSite(p, D, LS) \wedge \neg inPushFeed(p, LS)\} \\ P_2 &= \{\forall p : Patch(p) \wedge Document(D) \wedge Site(LS) \wedge Site(RS) \\ &\quad \wedge (LS \neq RS) \wedge onSite(p, D, RS) \wedge \neg inPullFeed(p, LS)\} \end{aligned}$$

It is interesting to notice that the SCHO model allows to use State Treemap metrics to calculate Palantir metrics. This was not possible without the formal ontology.

3.4.3 Concurrency Awareness Using SCHO

The patches which were made locally in parallel with the patches which were made remotely. If we know the causal relation between patches on a document

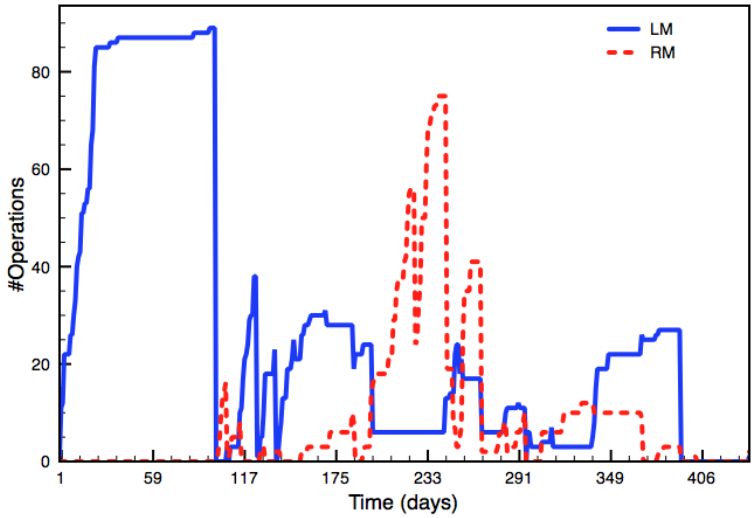


Figure 3.8: Divergence awareness results for gollum project (git)

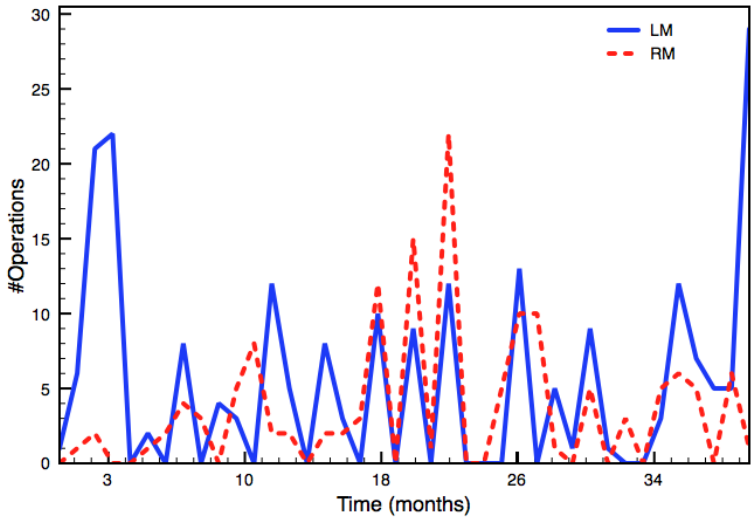


Figure 3.9: Divergence awareness results for mongoDB project (git)

we could know the concurrent patches. The multi-synchronous environment satisfies the causality which ensures that all the operations are ordered by a previous relation. This means that the operations will be executed in same order on every site. The history in this approach will be causal graph.

Definition 14 (Concurrent Modification (CM)) *The number of concurrent patches.*

$$
\begin{aligned}
CM \equiv & \exists D \exists S \exists P1 \exists P2 : \\
& Document(D) \wedge Site(S) \wedge hasDoc(S, D) \wedge \\
& onDoc(P1, D) \wedge onDoc(P2, D) \wedge \neg previous(P1, P2)
\end{aligned}
$$

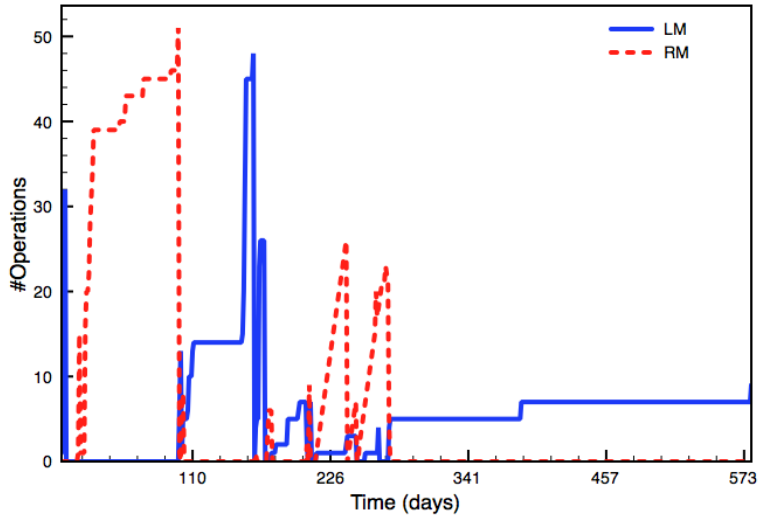


Figure 3.10: Divergence awareness results for AllTray project (Bazaar)

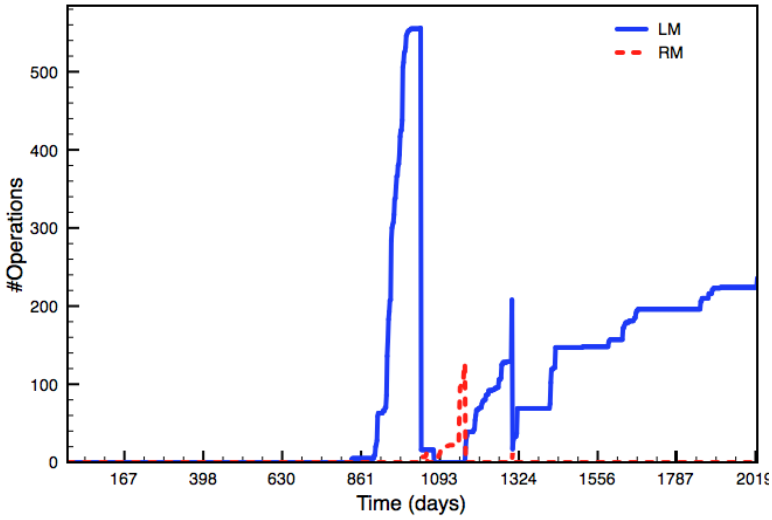


Figure 3.11: Divergence awareness results for Anewt project (Bazaar)

3.4.4 Validation

In order to validate our approach, we populated the *SCHO* ontology with causal history data from different DVCS. We used *git*, *Mercurial* and *Bazaar* repositories. These repositories have rich sets of data of different size that can be used to compute divergence awareness.

To use the DVCS data, first we had to inject the log data into a triple store to populate our ontology. We used the Jena TDB [8] triple store, then we implemented a parser called *dvcs2lod* [9]. *dvcs2lod* is responsible for the mapping between the concepts defined in the DVCS log and the *SCHO* ontology. It can handle *git*, *Mercurial* and *Bazaar* repositories.

We made the following assumptions when we parsed the causal history:

[8] http://jena.apache.org/documentation/tdb/index.html
[9] https://github.com/kmobayed/dvcs2lod

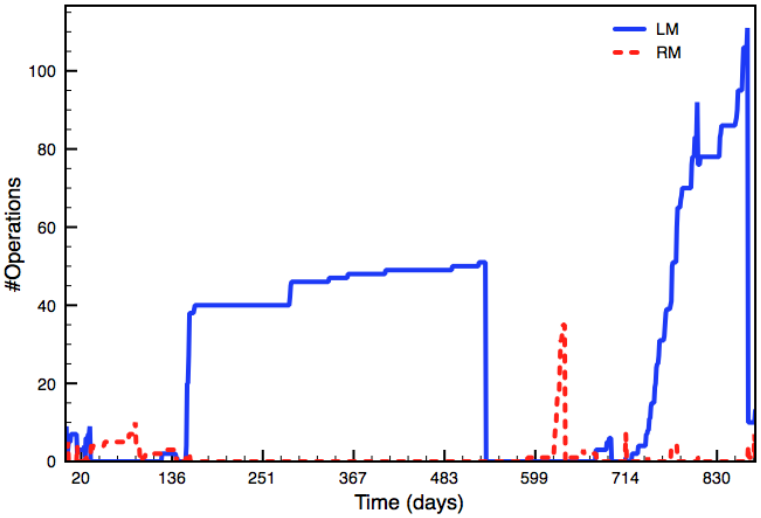


Figure 3.12: Divergence awareness results for hgview project (Mercurial)

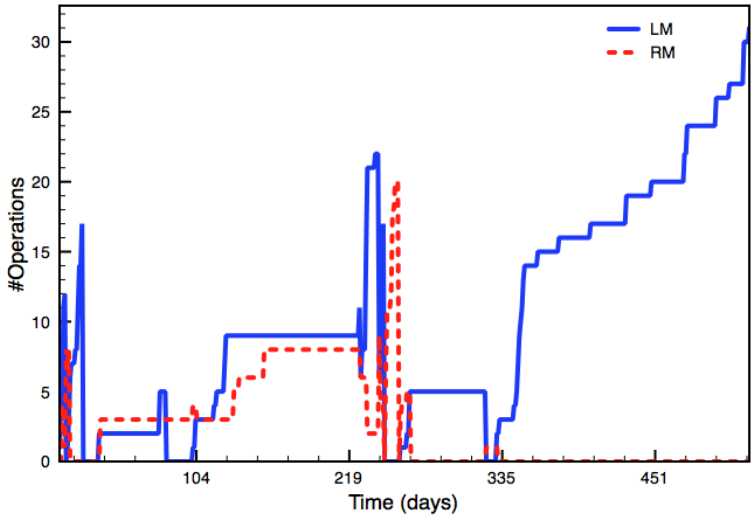


Figure 3.13: Divergence awareness results for Murky project (Mercurial)

- We considered the project as one shared object, so any changeset we find is a modification to this object.
- Whenever we find a branch in the history we create a PushFeed and the corresponding Site.
- Whenever we find a merge change set we create a PullFeed and the corresponding Site.
- Each site represents one user.

For simplicity, we will explain the above assumption on a sample project in *git* . Figure 3.4 shows the history log of this project.

For the sample project we will have: three Sites, two PullFeeds, two PushFeeds and thirteen ChangeSets. Figure 3.5 shows the populated ontology resulting from parsing the *git* history. Figure 3.6 shows the same RDF graph but without the concepts for the sake of clarity.

Based on the assumptions mentioned above we were able to find the different sites and the push/pull interactions between the sites as it is shown in Figure 3.6.

Now, we can use the metrics defined earlier which are generic divergence awareness metrics, and we can apply them on different open source projects that use different DVCS.

We use SPARQL [10] queries to calculate divergence awareness. For example the query shown in listing 3.10 returns the state *Remotely Modified* for a ChangeSet *CSid*.

```
SELECT ?pf WHERE {
      scho:CSid scho:inPullFeed ?pf .
      scho:CSid scho:date ?date .
      ?pf scho:hasPullHead ?CSHead
      scho:?CSHead scho:date ?headDate .
      NOT EXISTS scho:CSid scho:published "true".
      FILTER ( xsd:dateTime(?headDate) <= xsd:dateTime(?date) )
      }
```

Listing 3.10: Remotely Modified ChangeSet SPARQL Query

The following query returns the state *Published* for a ChangeSet *CSid*.

```
SELECT ?pf ?date WHERE {
      scho:CSid scho:inPushFeed ?pf .
      scho:CSid scho:date ?date .
      FILTER ( xsd:dateTime(?headDate) <= xsd:dateTime (?date))
      }
```

Listing 3.11: Published ChangeSet SPARQL Query

If a ChangeSet is not in one of these states i.e. not *Published* and not *Remotely Modified* then it will be in the state *Locally Modified* Table 3.1 shows the resulted states for State Treemap and Palantir for our sample project.

We plotted the corresponding divergence graph. Figure 3.7 shows the results we found for the sample *git* project. We can observe clearly the divergence and convergence phases.

Then we used real projects to validate the approach, such as: gollum [11], HgView [12], Murky [13], AllTray [14], Anewt [15], and MongoDB [16]. These projects use different DVCS such as git, Mercurial and Bazaar. Table 3.2 shows the details of each project and the execution time for populating the ontology with the causal history of these projects. In addition, we calculate the number of ChangeSets, Users, Merges and the number of triples generated based on the *SCHO* ontology.

Figures 3.8-3.13 show the results obtained after calculating the divergence awareness metrics on the selected projects. The *Y-axis* represents the number of changesets, while the *X-axis* represents the time. In each graph, we see the number of locally modified changesets (LM) and the number of the remotely

[10] http://www.w3.org/TR/rdf-sparql-query/
[11] https://github.com/github/gollum.git
[12] http://www.logilab.org/project/hgview/
[13] https://bitbucket.org/snej/murky/
[14] http://alltray.trausch.us/
[15] http://anewt.uwstopia.nl/
[16] http://www.mongodb.org/

modified changesets (RM) at a given time. We can clearly observe the periods of convergence and divergence. This clearly demonstrates that the same divergence awareness metrics can be represented and computed on data produced by different DVCS.

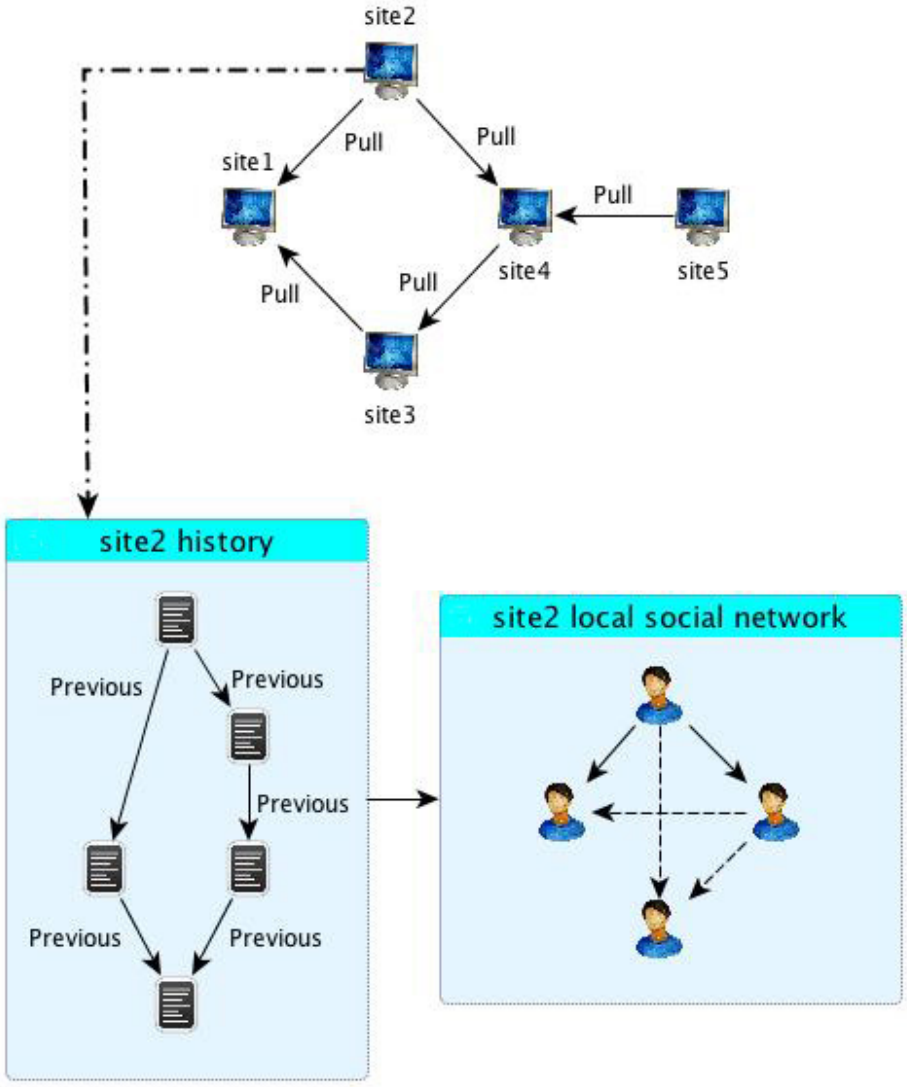


Figure 3.14: General approach illustration for extracting local social network

3.5 Local Social Network and Trust in SCHO

Decentralized multi-synchronous collaborative systems provide collaboration services without a dedicated service provider. For instance, Distributed Version Control Systems (DVCS) [2] demonstrated that it is possible to communicate and share data without the need for a collaboration provider. This allows to overcome problems related to the approach collaboration as a service (CaaS), where a service provider offers collaborative and social network services. The social service provider has access to all the data this raises privacy and censorship issues [33], the provider can exploit the whole social network

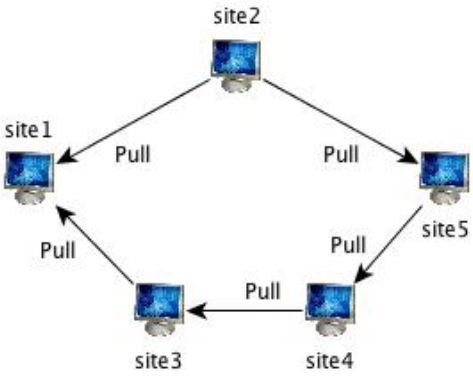


Figure 3.15: Collaboration scenario

relations and interactions among the users. However, the social relations provided by the social services are important to push further the collaboration between people. It is important to evaluate the location of actors in the network in order to understand networks and their participants; measuring the network location is essential. These measures give us insight into the various roles and groupings in a network: where are the clusters and who is in them, who is in the core of the network, and who is on the periphery.

Although decentralized systems provide the required collaborative services, they do not provide the social services offered by centralized systems. In these systems there is no central point with a global vision of the social network which is able to build and reflect the social relations among people.

Some approaches use private peer-to-peer networks [69] where the resources and the infrastructure are provided by the users participating in the network. Groove [55] is a groupware for collaborative editing which consists of isolated local networks. This system provides group-based network service which allows direct connections between the users of the group. Multi-synchronous semantic wiki [67] is another approach which allows direct connections between sites who know one another i.e. friend-to-friend network [11].

However, the collaboration model is very different from CaaS approach. With CaaS software, users mainly collaborate through read-write operations in one shared space provided by the collaboration provider. While in distributed social software, collaboration occurs by replicating shared data and synchronizing multiple workspaces continuously.

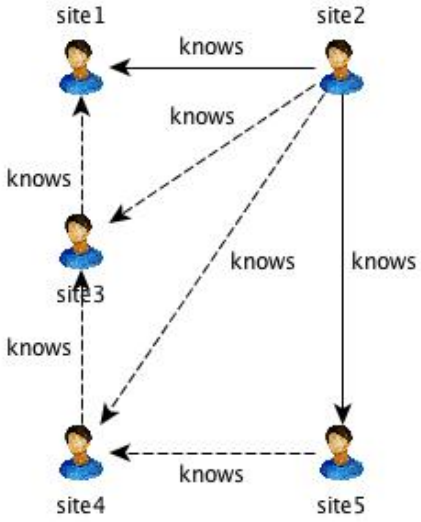
The previous model hides the social relations among sites participating in the network. Synchronizing workspace requires exchanging causal histories of operations. By analyzing the causal history on each site, we are able to reveal the social relations and reconstruct on each site a social network graph. So we can see our friends of friends.

This graph represents a local view of the social network and not the whole social network. Users can see their locations in their own local graphs. This approach can enrich the collaboration among the sites with new social services, while at the same time preserving the sites' privacy, since every site has a local vision of the network, and they do not rely on a service provider to maintain the social network.

The collaboration model in a multi-synchronous collaborative system hides

```
SCHO:Patch1
      SCHO:hasSite SCHO:Site1 ;
      SCHO:onPage SCHO:lesson1 .
SCHO:Patch2
      SCHO:hasSite SCHO:Site2 ;
      SCHO:onPage SCHO:lesson1 ;
      SCHO:previous SCHO:Patch1 .
SCHO:Patch3
      SCHO:hasSite SCHO:Site3 ;
      SCHO:onPage SCHO:lesson1 ;
      SCHO:previous SCHO:Patch1 .
SCHO:Patch4
      SCHO:hasSite SCHO:Site4 ;
      SCHO:onPage SCHO:lesson1 ;
      SCHO:previous SCHO:Patch3 .
SCHO:Patch5
      SCHO:hasSite SCHO:Site5 ;
      SCHO:onPage SCHO:lesson1 ;
      SCHO:previous SCHO:Patch4 .
SCHO:Patch6
      SCHO:hasSite SCHO:Site2 ;
      SCHO:onPage SCHO:lesson1 ;
      SCHO:previous SCHO:Patch5 .
```

(a) $Site_2$ causal history



(b) site2's local social network

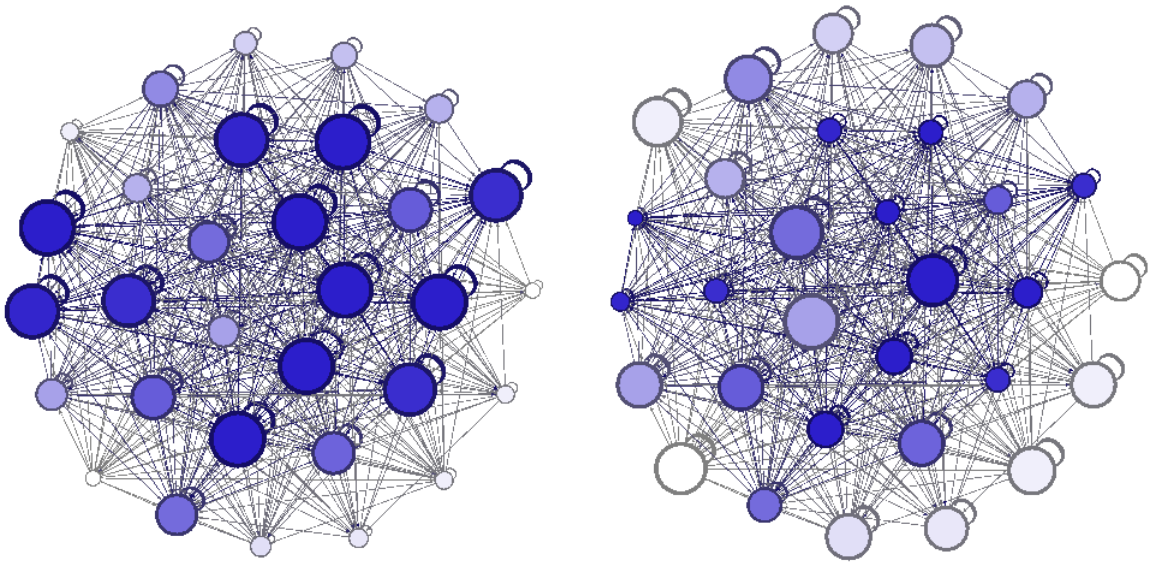
Figure 3.16: Social network extraction

the social relations among the sites. We will reveal these relations by transforming the *previous* relation between the patches into social relations among the sites participating in this network. Figure 3.14 shows an illustration of our approach. We show how we are going to discover the friend-of-friend relations. In this example we see the interaction between the sites participating in the network. It should be noted that in reality this interaction is not known by any site. This interaction generated a causal history on *site*2 for instance. This site has pulled from two sites only (*site*1 and *site*4) so it has a direct friend relation with these two sites. But it does not know the existence of the other sites in the network. By investigating its own causal history *site*2 will find patches generated on *site*3 and these patches has been pulled by *site*4. So now *site*2 knows the existence of *site*3 and the existence of a relation between *site*4 and *site*3. The deduced knowledge is represented by the dotted arrows in figure 3.14, this knowledge does not apply that *site*2 can pull from *site*3 since it does not have the capability required to pull from it.

In the next section, we will use the *SCHO* ontology to reconstruct the social network among the sites, visualize those relations and furthermore calculate the network centrality measures.

3.5.1 From Causal History to Social Relations

Although there is no direct friendship relation defined in a multi-synchronous collaborative system, users can manage their relations with others implicitly



(a) inDegree centrality (b) outDegree centrality

Figure 3.17: Degree centrality

by controlling from whom to accept modifications and to whom send or publish their modifications. This interaction is recorded in the causal history which is stored at each user's site. Each site participating in a multi-synchronous collaborative system keeps a complete causal history of all the operations it has received or generated locally.

According to *SCHO* ontology, a *patch* is a collection of operations generated at one site, and it is linked to other *patches* by the *previous* transitive relation. In order to be able to build the social relation among the sites, we extend *SCHO* ontology as follows:

- We define *hasSite* object property from Patch to Site as follows:

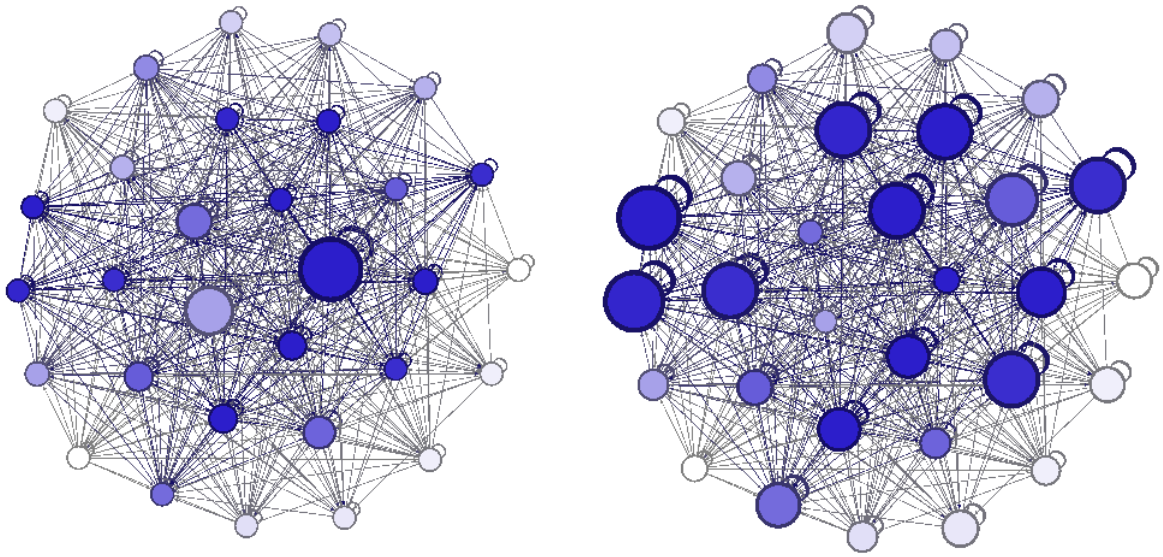
$$hasSite \equiv \exists(hasPull^{-1}).\exists(inPullFeed).\exists(hasPatch^{-1}).Patch$$

- We add an object property *knows* that checks if one site knows another site. We check if we have a *previous* relation between patch $P1$ generated on site $S1$ and patch $P2$ generated on site $S2$, if that is the case then this means that $S1$ pulled $P2$ from $S2$ which eventually means that $S1$ *knows* $S2$. This is calculated using the following inference rule:

$$knows(S1, S2) \sqsubseteq \exists P1.hasSite(P1, S1) \sqcap \exists P2.hasSite(P2, S2) \sqcap previous(P2, P1)$$

Where $P1, P2$ are both patches, and $S1, S2$ are both sites.

By taking the previous modifications into account we can rebuild a "FOAF:knows" relation [14] between the sites based on the push/pull feeds with a simple inference from the history using a semantic reasoner.



(a) Betweenness centrality (b) Closeness centrality

Figure 3.18: Betweenness and closeness centrality

The example in figure 3.15 shows the interaction between five sites. *site*1 creates a document, then *site*2 and *site*3 pull form *site*1; eventually *site*2 modifies the page; at the same time *site*3 makes some modifications too, then *site*4 pulls from *site*3; by its turn *site*4 modifies the page, then *site*5 pulls from *site*4 and modifies the page too. Finally *site*2 pulls from *site*5.

By investigating *site*2's history shown in figure 3.16(a). We can see some patches generated by *site*1, *site*3, *site*4 and *site*5 although it has only the capabilities from *site*1 and *site*5. Now *site*2 *knows* the existence of *site*3 and *site*4, moreover it can deduce the following: *site*3 *knows site*1, *site*4 *knows site*3 and *site*5 *knows site*4.

Now we can construct the local social graph of *site*2 as in figure 3.16(b). Where the solid-line arrows represent the direct relations between sites (by exchanging capabilities) while the dotted-line arrows represent the deduced relations which do not imply the ability to pull from these sites, since *site*2 has not the capabilities of these sites.

By applying this approach on all the sites, each site will have a local vision of his friend-of-friend network. We should take into consideration that the causal history keeps on growing and it will not delete previous entries. This means that the local vision at one site of the network keeps on growing as much as the site consumes from other sites. In the following section we present our validation and discuss the obtained results.

3.5.2 Validation

In order to compute the social network metrics with high confidence, we need a large dataset so we populated the SCHO ontology with the causal history of Mercurial repository for the Adium project [74], which follows the multi-synchronous collaboration model. The causal history that we analyzed is composed of 3128 patches (changesets in Mercurial terms) generated by 31 developers contributing to this project, over a period of 18 months. We were able

to reveal 588 *knows* relations between the developers of this project. We calculated the key parameters of the social network graph, in order to compare it with a real social network graph parameters. In the following we will explain the obtained results.

A graph is considered small-world, if its average clustering coefficient is significantly higher than a random graph constructed on the same vertex set, and if the graph diameter is much smaller than the order of the graph [4]. The calculated parameters show that the extracted social network graph follow the small-world hypothesis.

Social network parameter	Calculated value
Network diameter	3
Clustering coefficient	0.634
Graph density	0.632

We also computed the centrality measures identified by [30] and illustrated the results using Gephi [9]. First we calculate the degree centrality which considers nodes with higher degrees as more central, highlighting the local popularity of an actor in its neighborhood. The way that we used to generate the social network graph created a directed graph so we have to calculate the inDegree centrality as shown in figure 3.17(a) (the number of incoming connections) and outDegree centrality as shown in figure 3.17(b) (the number of outgoing connections).

Then we calculated the betweenness centrality which focuses on the capacity of a node to be an intermediary between any two other nodes. The results are illustrated in figure 3.18(a). This figure shows clearly that we have one special actor in the network (most probably the project coordinator).

Finally we calculated the closeness centrality of the graph nodes which represents the node capacity to be reached by any other node in the network. Figure 3.18(b) shows the calculated closeness centrality.

From the results shown above we clearly see that we have extracted the local social network and based on the data we used, we see that the social network characteristics follow the small-world model, and we can see that we have some important actors in our network.

We extended the collaborative services offered by multi-synchronous collaborative system with new social services by exploiting the causal history using semantic queries. Therefore, each site participating in this collaborative network can have a local view of its social network without the need for a third-party service provider, and using its own resources. We validated the approach using data from a software engineering application. We also found that the extracted network follows the small-world model.

3.6 Summary and Discussion

Existing divergence awareness systems such as State Treemap, Palantir, Edit Profile and Concurrency awareness define their own divergence metrics without a common formal definition. Metrics are coupled with the application and cannot be used outside their original scope. We proposed a unified ontology for conceptualizing multi-synchronous collaborative systems. This way we can compute all existing divergence awareness metrics as queries on a common ontology. These queries are performed on the causal history of the system,

on each participant workspace, this means that the result will be divergence awareness of the past. This raises the problem of how to compute divergence awareness in real-time? In order to answer this question we need to fulfill two requirements:

1. Who is participating in the system? This means we should provide and maintain a membership service.

2. What are the concurrent operations among the participants in the system? This means we need to access unpublished information on remote sites.

In the following chapter, we will extend the *SCHO* ontology to provide a membership service using semantic queries, and discover the collaboration network.

4

Network Discovery

Contents

4.1 Introduction

Multi-synchronous collaboration model does not define membership service, this implies that the collaboration networks are not navigable. Distributed Version Control Systems [2] (DVCS) are the most used systems that follow the multi-synchronous collaboration model. DVCS use a push-pull-clone paradigm described in section 2.1. In this paradigm, there is no standard way for a repository to advertise their push/pull relations, we can not navigate the social network generated by the users' interactions with each others. Actually the push/pull relations are private and not even published to others. This prevent the discovery of the DVCS social network. This is not a requirement for DVCS i.e. every user is free to pull changes from any source she wants, she is also free to keep this information private.

This issue has important consequences for software developers involved in software projects. For example: i) clustering can occur within the collaboration social network if one developer shuts down her repository; ii) estimating the global activity of the collaboration social network is also important for project management, awareness and coordination.

Navigability of collaboration social network can be achieved if all DVCS repositories for a software project are hosted within a single software forge such as github (`https://github.com`), launchpad (`https://launchpad.net`), and

sourceforge (https://sf.net). github allows navigation between git repositories hosted on github. Of course, this approach is very restrictive i.e. navigability of collaboration social networks should not rely on service providers.

Another approach is to use semantic web technologies and transform any DVCS into a social semantic web tool. Semantic web technologies are inherently distributed and offer strong support for interoperability.

We propose a lightweight ontology *SCHO+*. *SCHO+* is based on the observation that existing DVCS follow the optimistic replication model [71]. *SCHO+* conceptualizes causal histories and push/pull relations. Based on *SCHO+*, we give the opportunity to actors to extract information from their local DVCS repositories and generate RDF datasets. These data are clearly targeted for the linked data and reuse FOAF [14]/DOAP [25] vocabularies. Next, each actor can perform queries on the collaboration social network using Link Traversal Based Query Execution [41]. In order to validate collaboration social network requests, we experiment the discovery of the collaboration network for a group of developers.

A motivating example is shown in figure 4.1, outlines the problems of interoperability and membership in DVCS.

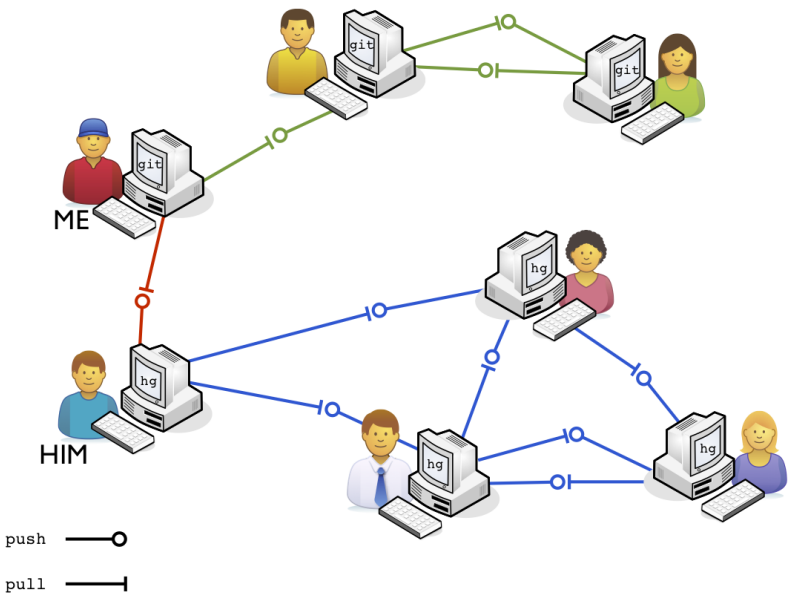


Figure 4.1: Multi-synchronous Collaboration using different DVCS

In this scenario, we have different groups of developers collaborating to build a software. The green links are made using git push/pull operations, while the blue links are made using Mercurial push/pull operations. "HIM" is the Mercurial team leader and "ME" is the git team leader. There is a red link between "HIM" and "ME" that is made using the hg-git [1] tool. From this scenario we see that it is crucial for "HIM" to maintain a connection with the git group, to advance the collaboration. This connection is currently assured by "ME", but what if "ME" goes down? The other way around is also valid, since "ME" needs to maintain a connection with the Mercurial group. This clearly shows the need for a membership and network discovery service. So

[1] http://hg-git.github.com/

in case of a failure, collaborators can switch to other ones to maintain their collaboration activity.

The push/pull interactions in the DVCS are not recorded in standard format. Currently there is no way to run queries to explore the collaboration social network. Or even to calculate a common metric to capture the network activity.

4.2 Distributed Version Control Systems

Version control systems (VCS) [2] such as *RCS* [2], *CVS* [3] and *Subversion* [4] are used by software developers to maintain source code, documentation and configuration files for their projects. *RCS* is suitable for single-user scenarios, while *CVS* and *Subversion* are based on centralized approaches. Newer version control systems such as *git* (http://git-scm.com/), *Mercurial* (http://www.selenic.com/mercurial/), *Bazaar* (http://bazaar.canonical.com/) and *Darcs* (http://www.darcs.net/) are distributed version control systems (DVCS) [2]. The main characteristics of DVCS compared to traditional VCS are decentralization and autonomous participants. DVCS are one of the most used systems that follows the multi-synchronous collaboration model.

DVCS are social tools largely used in open source software development. They allow huge community of developers to coordinate and maintain software such as the Linux kernel project [5] and Mozilla Firefox [6].

Every new developer involved in a software project can set up her own code repository by cloning an existing one and start contributing by advertising new updates. We will refer to this model of collaboration as Push-Pull-Clone (PPC). The PPC collaboration model generates networks of DVCS repositories linked by push/pull relations. These networks are very similar to social networks where a user can follow messages of other users. The main difference comes from the nature of exchanged messages i.e. in a DVCS, messages contain operations that can be applied on local files or directories.

Although existing DVCS rely on the PPC model and have the same basic concepts, they suffer from interoperability problems. Working using the PPC model creates a network of collaborators, but there is no standard way for the DVCS to publish the push/pull relationships. Nowadays, this network is hidden and we can not run any query on it. Discovering the collaboration relations is important to push further the collaboration between people. It is also important to evaluate the location of actors in the network [30]. This will give us insight on the collaboration activity which is crucial for project management.

The correctness of the system is defined by properties such as causality, eventual consistency, and intention preservation [81]. All DVCS ensure at least causal consistency [51]. A system ensures causal consistency if all sites execute the same set of operation in the same order of generation. The traditional way to implement causal consistency is to implement a causal reception i.e.

[2] http://www.gnu.org/software/rcs/
[3] http://savannah.nongnu.org/projects/cvs
[4] http://subversion.apache.org/
[5] http://www.kernel.org/
[6] http://www.mozilla.com/

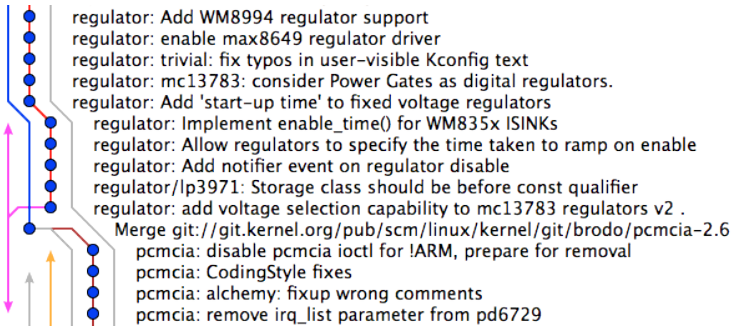


Figure 4.2: Causal history in git

an operation is delivered to the local processes if all causal operations have been delivered before. In distributed systems, causal reception is implemented traditionally with vector clocks [54]. The eventual consistency property [48] ensures that all replicas are identical when the system is idle. Thomas write rule [48] was the first algorithm to ensure eventual consistency in duplicated databases. However, Thomas write rule requires the knowledge of the number of participants (in order to provide a safe garbage collection scheme).

DVCS follow the optimistic replication model:

1. Each repository is a site where objects i.e. files and directories are replicated.

2. Object can be modified anytime, anywhere by applying operations. In DVCS, this is achieved by generating "commit objects" that can be interpreted as a set of operations updating several objects.

3. Operations are broadcasted to other sites. In DVCS, broadcast is achieved through push/pull operations. This can be interpreted as an anti-entropy mechanism that is part of epidemic protocols [19]. Anti-entropy protocols ensure causal delivery of operations.

4. Causal relationships are not represented with vector clocks that require join and leave procedure, but with explicit "previous relations" between "commit objects". These relations can be observed in the graph of figure 4.2. This figure presents a fragment of the causal history of the Linux kernel.

5. DVCS ensure at least causal consistency i.e. all repositories execute the same operations respecting the same causal order. As causal order is a partial order, it does not mean that all sites have the same execution history, but they will all converge to the same causal graph.

4.3 Linking SCHO to the Linked Open Data

The Shared Causal History Ontology (SCHO) [5] is an ontology for sharing and managing causal history. SCHO ontology defines all the basic concepts common to DVCS such as ChangeSet, Patch, Previous, Operation, etc. It also defines more precise concepts such as PullFeed and PushFeed to support the PPC model. The existence of a PullFeed on *Site1* that consumes operations

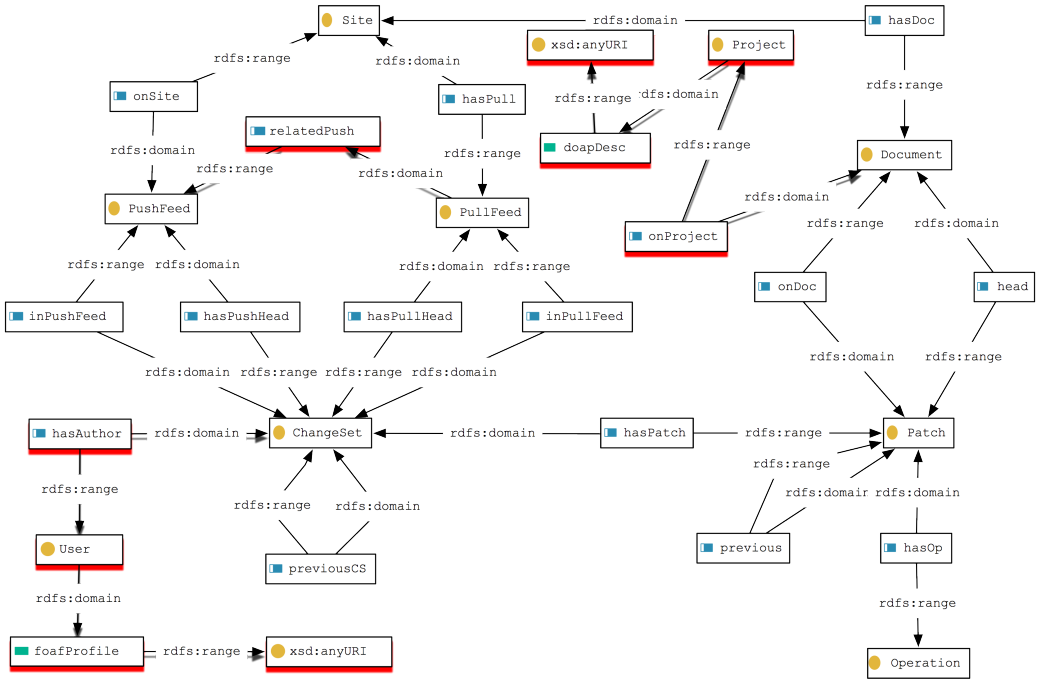


Figure 4.3: Shared Causal History Ontology Extension

from a PushFeed on *Site2* can be interpreted as *follow* relation between the two sites, i.e. *Site1 follow Site2*.

We will extend the *SCHO* ontology in order to link participants and objects by using FOAF/DOAP vocabularies and links the DVCS community to the LOD cloud. The extended ontology is called: *SCHO+*. We add a new class *User* presented in listing 4.1. We link it to the FOAF profile of changesets' authors using an owl:DatatypeProperty *foafProfile*. We add a new class *Project* presented in listing 4.2. We link it to the DOAP description for each project using an owl:DatatypeProperty *doapDesc*. We also add a new owl:ObjectProperty *relatedPush* presented in listing 4.3. This owl:ObjectProperty links the *PullFeed* to its corresponding *PushFeed*. We also modify the CreatePull algorithm to take this into consideration, as it is shown in listing 4.4. This will enable us to extract the *follow* relation between the sites that generated these feeds. We will use this *follow* relation to discover the PPC social network between the different sites. Figure 4.3 shows the *SCHO+* ontology[7].

```
<owl:Class rdf:about="#User">
        <rdfs:subClassOf rdf:resource="&owl;Thing"/>
</owl:Class>
<owl:DatatypeProperty rdf:about="#foafProfile">
        <rdfs:domain rdf:resource="#User"/>
        <rdfs:range rdf:resource="&xsd;anyURI"/>
</owl:DatatypeProperty>
```

Listing 4.1: New class User

[7] http://kolflow.univ-nantes.fr/mediawiki/images/scho+.owl

```
<owl:Class rdf:about="#Project">
       <rdfs:subClassOf rdf:resource="&owl;Thing"/>
</owl:Class>
<owl:DatatypeProperty rdf:about="#doapD">
       <rdfs:domain rdf:resource="#Project"/>
       <rdfs:range rdf:resource="&xsd;anyURI"/>
</owl:DatatypeProperty>
```

Listing 4.2: New class Project

```
<owl:ObjectProperty rdf:about="#relatedPush">
       <rdfs:range rdf:resource="#PullFeed"/>
       <rdfs:domain rdf:resource="#PushFeed"/>
</owl:ObjectProperty>
```

Listing 4.3: Related Push Property

```
createPull (int pullID, int pushID)
{
       PullFeed PF= new PullFeed(pullID);
       relatedPush(pullID, pushID);
       hasPull(site, PF);
       Pull(PF);
}
```

Listing 4.4: createPull operation

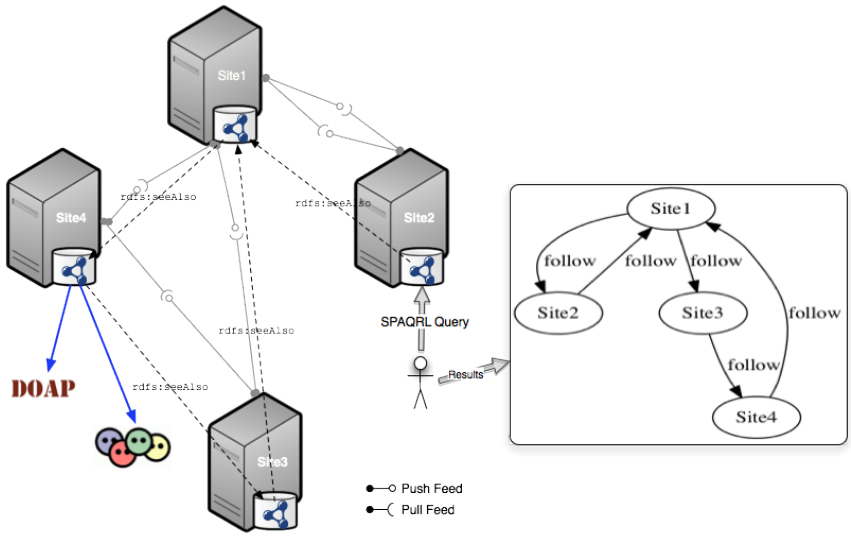


Figure 4.4: Push/Pull Network

Generating RDF [8] triples and linking the RDF graphs of the different sites will enable us to query and discover the PPC social network as shown in figure 4.4. In this scenario, four sites are using DVCS and are connected to each other with push/pull links. These links can be seen as an ad-hoc P2P network. In this scenario :

- $Site_1$ and $Site_2$ are engaged in a push/pull from each other. This means that $Site_1$ is pushing its modifications to $Site_2$ and $Site_2$ is accepting these modifications, and vice versa.

[8] http://www.w3.org/TR/rdf-primer/

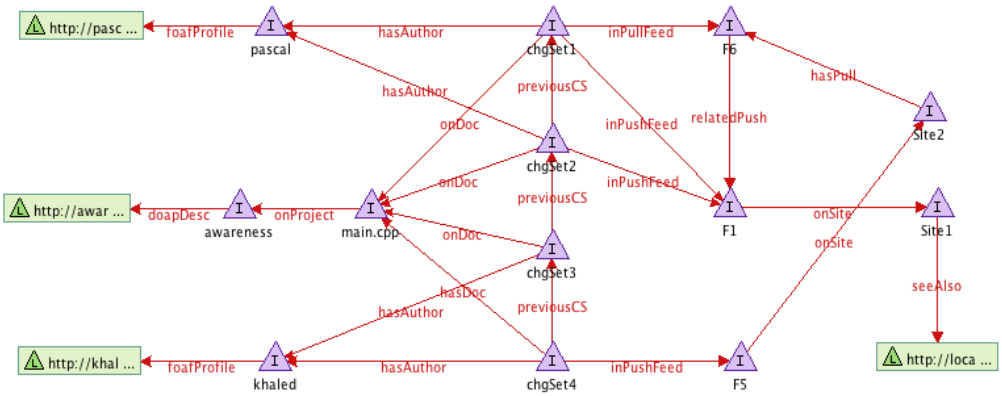


Figure 4.5: $Site_2$ RDF graph

- $Site_3$ pushes modifications to $Site_4$ and pulls from $Site_1$.

- $Site_4$ pushes modifications to $Site_1$ and pulls from $Site_3$.

The push/pull interactions in the DVCS generate triples based on a common ontology at each site. These triples are stored in an RDF file that have an accessible URI. This way a user on a given site can run a distributed SPARQL query to explore the PPC social network. Or she can run a divergence awareness metric query to capture the network activity. A user can link her *foaf:profile* and the project description *doap:project*. So her data will be available to the whole LOD community.

We expect each DVCS user to run a script that will publish some information about the current state of his personal workspace. This information will be published as an RDF file conform to the *SCHO+* ontology. Next each user can run semantic queries relying on Link Traversal Based Query Execution.

The advantages of using *SCHO+* ontology are:

- First we have a unified minimal ontology for representing and managing the shared causal history of any DVCS. This will make it easier to develop universal tools and plug-ins for any DVCS software that adopts this ontology for managing its log. So we can run queries directly on any DVCS system that uses *SCHO+* ontology without the need to import/export histories between different DVCS tools.

- Furthermore we can have generic analysis tools which can be run over this log to discover the underlying dynamics of the corresponding network. We have also linked the ontology to the LOD using the DOAP and FOAF ontologies. This will permit to link the DVCS communities with LOD and will give them a higher visibility. In order to be included in the analysis and statistics done on the LOD.

Figure 4.5 shows the corresponding RDF graph of $Site_2$ from the scenario presented in figure 4.4. In the following section, we detail the queries that allow each user to extract the PPC social network.

```
:Site1    a scho:Site;
    scho:hasPull :F2,
          :F4 .
:Site2    a scho:Site;
    rdfs:seeAlso <site2_RDF_URI> .
:Site3    a scho:Site;
    rdfs:seeAlso <site3_RDF_URI> .
:Site4    a scho:Site;
    rdfs:seeAlso <site4_RDF_URI> .
:F1    a scho:PushFeed;
    scho:onSite :Site1 .
:F2    a scho:PullFeed;
    scho:relatedPush :F5 .
:F3    a scho:PushFeed;
    scho:onSite :Site1 .
:F4    a scho:PullFeed;
    scho:relatedPush :F8 .
:F5    a scho:PushFeed;
    scho:onSite :Site2 .
:F8    a scho:PushFeed;
    scho:onSite :Site4 .
```

(a) $Site_1$ RDF file

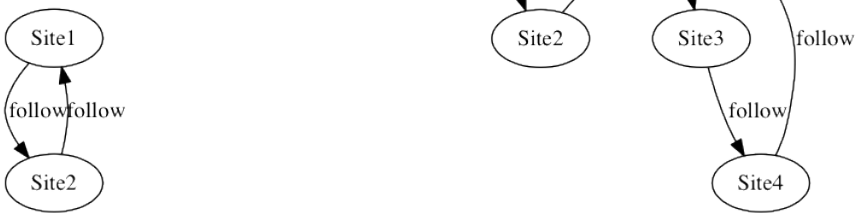
```
:Site2    a scho:Site;
    scho:hasPull :F6 .
:Site1    a scho:Site;
    rdfs:seeAlso <site1_RDF_URI> .
:F5    a scho:PushFeed;
    scho:onSite :Site2 .
:F6    a scho:PullFeed;
    scho:relatedPush :F1 .
:F1    a scho:PushFeed;
    scho:onSite :Site1 .
```

(b) $Site_2$ RDF file

Figure 4.6: Scenario example RDF files

4.4 Validation

Linking the RDF graphs using the *SCHO+* ontology will allow the discovery of the PPC social network without the need for a centralized node or a social service provider. The social service provider has access to all the data which raises privacy and censorship issues [33], since it can exploit the whole PPC social network relations and interactions among the users. In order to overcome these issues, new decentralized approaches were proposed. They provide collaborative services without a dedicated service provider. Users can create their own collaborative network and share the collaborative services offered by the system using their own resources. If it is easy for a centralized node to extract the complete social network graph from the observed interactions. Obtaining social network information in the distributed approach is more challenging. In fact, the distributed approach is designed to protect privacy of users and thus



(a) $Site_2$ PPC social network without linking the RDF files

(b) $Site_2$ PPC social network with linking the RDF files

Figure 4.7: Network discovery

makes extracting the whole social network difficult.

We will show how linking the RDF graphs would make the PPC social network discovery easier. On one hand, this social network is independent of the project that the user is working on. On the other hand, this social network is also independent of the used DVCS. i.e. the we will have a higher level of abstraction for this PPC social network. We add a semantic annotation using the *rdfs:seeAlso* property to each site. This annotation will include the URI of the RDF file of each site.

```
SELECT DISTINCT ?site1 ?site2 WHERE {
       ?site1 a scho:Site .
       ?site2 a scho:Site .
       ?pull a scho:PullFeed .
       ?push a scho:PushFeed .
       ?pull scho:relatedPush ?push .
       ?push scho:onSite ?site1 .
       ?site2 scho:hasPull ?pull .
       FILTER (?site1 != ?site2)
       }
```

Listing 4.5: SPARQL Query for Network Discovery

In fact, using the *SCHO+* ontology renders discovering the network, no more than executing a SPARQL query. We will use the Link Traversal Based Query Execution approach [41] to execute this query. The advantages of this approach are: There is no need to know all the data sources in advance, the queried data will be up-to-date, and it is independent of the existence of SPARQL endpoints provided by the data sources. Listing 4.5 shows this query.

We will take the previous example presented in figure 4.4. In this example, we have $Site_2$ collaborates with $Site_1$ but it has no direct knowledge about the whole network. Figure 4.6 shows snapshots of the RDF files present on $Site_1$ and $Site_2$. We will extract the PPC social network using the SPARQL query in Listing 4.5. This query will give us a list of sites that have a collaboration link among them. We visualize the output using *graphviz* [9] graph visualization software. First, we run the query over $Site_2$ RDF file without the *rdfs:seeAlso*

[9] http://www.graphviz.org/

annotation, see figure 4.7a. Next, we run the same query over $Site_2$ RDF file but this time with the *rdfs:seeAlso* annotation, see figure 4.7b.

We evaluate the performance of the Link Traversal Based Query Execution approach [41] by simulating various network settings. We generate different configurations of multi-synchronous collaboration networks where sites are linked according to different graph models. We made the hypothesis that there is no isolated sites in the system. We used the Erdös-Rényi [27] random graph model and Barabási-Albert [1] scale-free networks that better represents social graphs. Table 4.1 shows the different characteristics of the generated networks. We generated graphs up to 500 sites. The third column represents the number of nodes in the network, the fourth column represents the average number of edges per node, and the last column represents the network discovery execution time. We used Cytoscape[10] to generate the different graphs. Simulations demonstrate the poor performance of the Link Traversal Based Query Execution approach. This approach does not scale for large distributed collaborative systems.

Setup	Network model	#Nodes	#Edges	Query execution time (ms)
1	Erdos-Renyi	100	10	77296
2	Erdos-Renyi	100	50	487289
3	Erdos-Renyi	500	250	12083689
4	Barabasi-Albert	100	10	18837
5	Barabasi-Albert	500	10	1849033

Table 4.1: Network discovery time results for different network setups

4.5 Summary and Discussion

In this chapter we proposed a membership service for multi-synchronous collaborative systems which is essential for divergence awareness computation. The membership service relies on the the Link Traversal Query Based Execution approach. The advantage of this approach that it does not require a beforehand knowledge of the sites that it will seek for getting the data to evaluate the query result, which is compatible with the multi-synchronous collaboration model. But after running simulations for evaluating the approach, we found out an extremely poor performance, this approach does not scale and makes it impossible to rely on it for computing divergence awareness. To find an efficient algorithm to compute divergence awareness we need to formally define what is divergence awareness for a group of collaborators. Divergence awareness has not been formalized before, all existing proposals are ad-hoc implementations for certain collaboration software. Existing divergence metrics rely on different interpretation of editing distance between copies in the system. Computing divergence awareness is very challenging, it requires to query remote states of all participants and next aggregates information to compute a global metric. This raises severe problems of performances, reliability and dynamism of participants. In the following chapter we formally define divergence

[10] An Open Source Platform for Complex Network Analysis and Visualization:http://www.cytoscape.org/

awareness using the GroupDiv model. Then we propose efficient algorithms to calculate group divergence awareness for the past and in realtime.

5

GroupDiv: Group Divergence Awareness Formal Model

Contents

5.1 Introduction

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 [46]. The overhead of solving conflicts introduced by concurrent modifications can overwhelm the expected gain [17, 64, 66]. Divergence awareness [58] makes participants conscious of the quantity and the location of divergence in shared objects. Participants are informed about potential risk of future conflicts. Divergence awareness answers the following questions: is there any divergence? With whom? Where? And how much? Divergence awareness is an implicit

coordination mechanism [39, 34], 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 [57], Operational Transformation Divergence [58], Palantir [73], Edit Profile [63], Concurrent modifications [3], and Crystal [15].

Different approaches exist for computing divergence: estimating the size of conflicts [58], estimating the difference between users' copies and a reference copy [57], or estimating divergence according to multiple copies of reference [15]. Some systems only consider published operations [63], therefore, we can talk about divergence awareness of the past. Others consider unpublished operations [46], in this case, we have real-time divergence awareness. 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, it is more difficult to understand how they really quantify divergence. Most of the metrics estimates some editing distance between users' workspaces and a copy of reference. They do not really try to quantify the group divergence. In this chapter, we focus on the quantification of the divergence of a group. A group divergence metric makes all members of the group aware of the minimal distance of the group to reach the next potential convergence point. 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. This way, a reference copy is defined virtually according to the current global state of the group.

Once a group divergence metric is defined, a second issue arises concerning the computation of this metric. Group divergence metrics require some global knowledge about the state of system. Multi-synchronous collaborative systems can be centralized such as CVS or Dropbox. In this case, divergence information can be exchanged through a central server or simple notification systems as in [57, 73]. It can also be fully distributed as in distributed version control systems or P2P wikis [67] and organized as a social network. In this case, the number and the list of sites are unknown which makes any group divergence metric computation more complex as described in [7].

In this chapter, we define a simple formal model for divergence metrics and we propose an original group divergence metric as the number of operations to integrate by the group to reach a convergence state. We propose an algorithm to compute this group divergence metric for causal histories. The metric is expressed as semantic queries on a conceptualization of our formal model. We validate this approach by computing group divergence metric on real histories extracted from different distributed version control systems. We also define a distributed algorithm to compute group divergence metric efficiently using gossiping protocols in a fully decentralized network. We validate the distributed computation of divergence metrics with simulations on a peer-to-peer network.

5.2 GroupDiv Definition

We observed that existing multi-synchronous collaborative systems behave like optimistic replication systems [71]. 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 [23]. 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. An operation can be decorated with meta-data such as author, date, operation identifier, etc.

2. It is broadcasted to all other sites. The broadcast is supposed to be 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 model. Integration relies on merge algorithms such as those used in operational transformation[81]. 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 [51], 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 history of operations. These operations are partially ordered using causal relations. The local history is "grow only", we call it the causal history of a site.
- 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.

- Building 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.

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 $Site_{id}$, and it maintains a logical clock $Clk_{Site_{id}}$ [54, 68] that increments when the site generates a new operation. An operation op is uniquely identified by the pair $(Site_{id}, Clk_{Site_{id}})$, where $Site_{id}$ is the generating site identifier and $Clk_{Site_{id}}$ is its logical clock when it generated the operation. We note $\mathbb{H}$ as the global causal history of the system, it is the set of all existing operations in the system at the computation time.

An operation op_2 is causally dependent on op_1 if they are related by the happened-before relation [51, 81].

Definition 15 (happened-before $\rightarrow$) *Given $op_1, op_2 \in \mathbb{H}$, generated respectively at sites: $Site_1$ and $Site_2$: $op_1 \rightarrow op_2$ iff (i) $Site_1 = Site_2$ and the generation of op_1 happened before the generation of op_2; or (ii) $Site_1 \neq Site_2$ and the execution of op_1 at $Site_2$ happened before the generation of op_2, or (iii) $\exists op_3 \in \mathbb{H} : (op_1 \rightarrow op_3) \wedge (op_3 \rightarrow op_2)$*

The happened-before relation defines a partial order on the set of the operations in the system $\mathbb{H}$. The happened-before relation is transitive, irreflexive and antisymmetric. From the happened-before definition, we can define concurrent operations as:

Definition 16 (Concurrent operations $\|$) $\forall op_1, op_2 \in \mathbb{H} : op_1 \parallel op_2 \Leftrightarrow \neg(op_1 \rightarrow op_2) \wedge \neg(op_2 \rightarrow op_1)$

We associate a local causal history H_S with every site S in the system. This corresponds to all operations generated and received by the site S at divergence awareness computation time. The definition implies that if an operation belongs to the local history, then all operations that happened before this operation belongs also to the local history. Consequently, the history is complete.

Definition 17 (Local Causal history) $\forall S \in \mathbb{S} : \exists H_S \subseteq \mathbb{H}$ *such that* $\forall op \in H_S, \{op_x : op_x \rightarrow op\} \subseteq H_S$

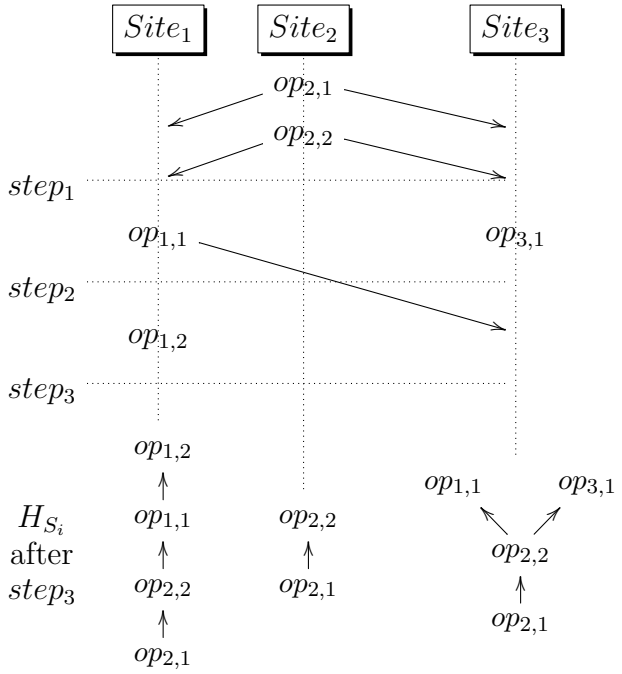


Figure 5.1: Multi-synchronous collaboration scenario

The scenario in figure 5.1 shows three sites collaborating using multi-synchronous collaboration model. The local causal history H_{S_i} for each site appears at the end of the figure. Every site maintains a logical clock Clk_{S_i} and a causal history H_{S_i}. The causal history is modeled as a directed acyclic graph with the operations as nodes and the happened-before relations as arcs. At the beginning $Site_2$ generates two operations $op_{2,1}$ and $op_{2,2}$, then it broadcasts these operations to the other sites ($step_1$). $Site_1$ and $Site_3$ generate $op_{1,1}$ and $op_{3,1}$ respectively after receiving $Site_2$'s operations ($step_2$). $Site_1$ sends its operation to $Site_3$, then it generates new operation $op_{1,2}$ ($step_3$). After receiving the operation $op_{1,1}$, $Site_3$ can deduce that: $op_{2,1} \rightarrow op_{2,2}$, $op_{2,2} \rightarrow op_{1,1}$, $op_{2,2} \rightarrow op_{3,1}$, $op_{1,1} \parallel op_{3,1}$ and by transitivity $op_{2,1} \rightarrow op_{1,1}$, $op_{2,1} \rightarrow op_{3,1}$.

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

Definition 18 (H_S is incremental) *When a site S receives or generates a new operation op then $H_S = \overline{H_S} \cup \{op\}$; $\overline{H_S} \subseteq H_S \subseteq \mathbb{H}$*

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. From H_{max} we can deduce the total number of unique operations in the system.

Definition 19 (Maximal causal history in the system)

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

Definition 20 (The total number of operations in the system) $op_{tot} = |H_{max}|$

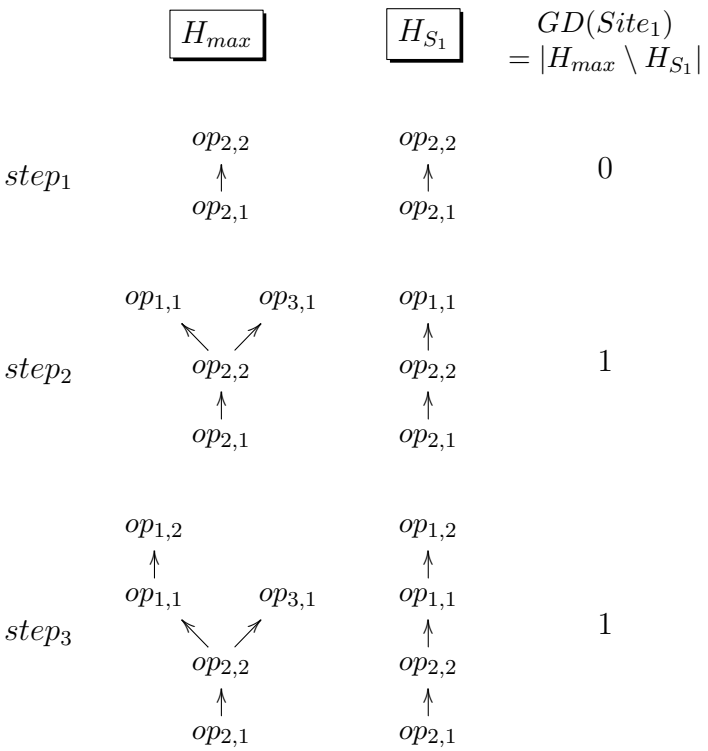


Figure 5.2: $GD(Site_1)$ computation at each step of the scenario of figure 5.1

We define the global divergence of a site S_i as the the sum of operations in H_{max} that are not in S_i's causal history, $GD(S_i)$. It 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}.

Definition 21 (Global divergence of a site) $GD(S_i) = |H_{max} \setminus H_{S_i}|$

We can compute $GD(Site_1)$ for the scenario of figure 5.1. At $step_1$, H_{max} is equal to H_{S_1}, so $GD(Site_1) = 0$. At $step_2$ and $step_3$, only one operation is missing for $Site_1$, so $GD(Site_1) = 1$.

The group divergence GD_{tot} is the sum of operations in H_{max} that are not in H_{S_i} of every site S_i in the system. The system is idle when divergence is null i.e. when all H_{S_i} are equal to H_{max}.

Definition 22 (Group divergence)

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

Figure 5.3 shows the results of calculating H_{max} and the group divergence GD_{tot}. For example, at step 3, every user knows that the group divergence is $GD_{tot} = 5$. This represents the distance for next potential convergence state. Every user knows her own contribution to this distance. $Site_1$ has to integrate $op_{3,1}$, $Site_2$ has to integrate $op_{3,1}$, $op_{1,1}$ and $op_{1,2}$, $Site_3$ has to integrate $op_{1,2}$.

GD_{tot} as defined in definition 22 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 as follows:

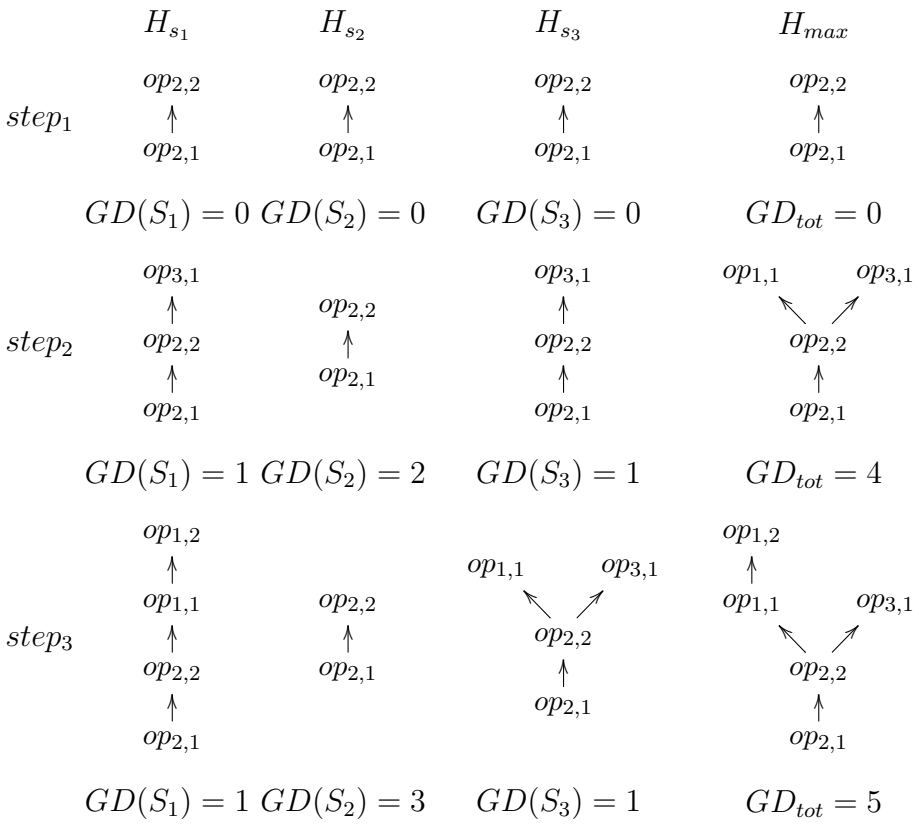


Figure 5.3: Max causal history and group divergence for three sites

Definition 23 (Aggregate global divergence for a site)

$$op_{tot} = \sum_{k=1}^{N} Clk_{S_k} : \forall S_k \in \mathbb{S}$$
$$AGD(S_i) = op_{tot} - |H_{S_i}|$$

Consequently, we can rewrite the computation of group divergence GD_{tot} in the system as follows:

Definition 24 (Aggregate group divergence)

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

This definition only requires aggregation functions for computation. To better understand GD_{tot} definition, 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 25 (Divergence between two sites) $\nabla : \mathbb{S} \times \mathbb{S} \rightarrow \mathbb{N}$, $\nabla(S_1, S_2) = |(H_{S_1} \setminus H_{S_2}) \cup (H_{S_2} \setminus H_{S_1})|$

Definition 26 (∇ is commutative) $\forall S_1, S_2 \in \mathbb{S} : \nabla(S_1, S_2) = \nabla(S_2, S_1)$

We calculate the divergence between the sites in the scenario in figure 5.1, the results are shown in the table 5.1.

Step	$\nabla(Site_1, Site_2)$	$\nabla(Site_1, Site_3)$	$\nabla(Site_2, Site_3)$
$step_1$	0	0	0
$step_2$	1	2	1
$step_3$	2	2	2

Table 5.1: Divergence between each two sites in the system

In order to calculate the group divergence, it is incorrect to only rely on ∇. As it is shown in figure 5.1, $Site_3$ has consumed $op_{1,1}$ from $Site_1$. $op_{1,1}$ will be calculated twice: 1) in $\nabla(Site_1, Site_2)$ and 2) in $\nabla(Site_2, Site_3)$ 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.

5.3 Computing Divergence Awareness on Causal Histories

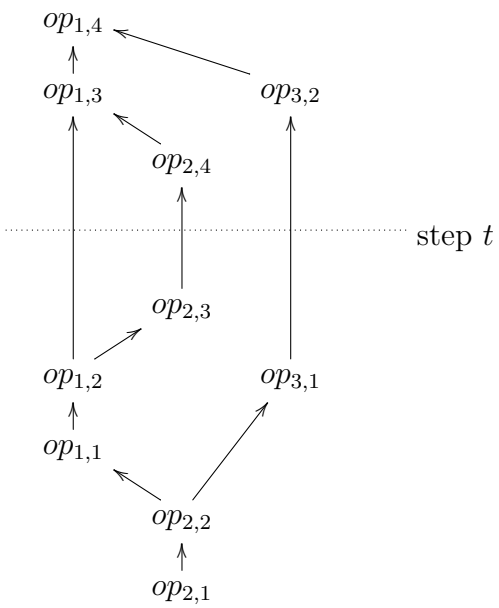


Figure 5.4: A causal history

In this section, we show how we computed GD_{tot} on real data extracted from different software projects managed by different distributed version control systems (DVCS). First, we built extractors to populate the SCHO ontology defined in [7, 5], SCHO conceptualizes causal histories. This demonstrates that different multi-synchronous collaborative systems rely on common abstraction and this abstraction is enough to compute group divergence awareness, as we will see later. Next, we designed an algorithm that performs a bottom-up scan

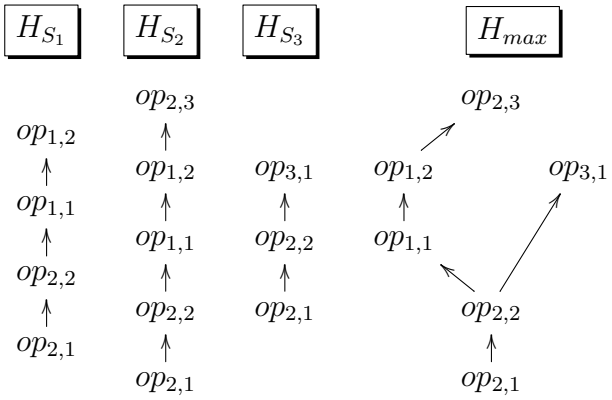


Figure 5.5: Extraction of H_{S_i} and H_{max} at step t from the causal history described in figure 5.4

of the causal histories in order to compute GD_{tot}. This algorithm allows to replay the evolution of GD_{tot} on existing causal histories. This helps the analysis of project history in terms of frequency of convergence and the maximal amplitude of divergence.

Computing group divergence metric on existing causal histories requires *a posteriori* an interpretation of a causal history. We have to determine what is H_{max} and what are H_{S_i}. Figure 5.4 presents a causal history from which group divergence metrics will be computed. By hypothesis, we consider that all sites are present in this history. The algorithm for computing divergence awareness metric is detailed in listing 5.1. The algorithm has three steps:

1. Starting from the bottom of causal history, we have to decide when to perform computation. We choose a cut point (step t in figure 5.4). Different strategies can be applied: operations can be annotated with received timestamp and slicing can be done on intervals, or the causal history is considered as a lattice and every slice corresponds to a level in the lattice. In this algorithm, we choose to use operations' timestamps to determine the cut.

2. Once the cut is determined (step t in figure 5.4), we have to determine the set of H_{S_i} and H_{max}. H_{max} is composed of all operations from the bottom to step t. Next, we have to determine the number of sites involved in H_{max}. As operations are uniquely identified by the pair $(Site_{id}, Clk_{Site_{id}})$, N is the number of different $Site_{id}$ present in H_{max}. H_{S_i} contains the last operation produced by $Site_i$: $op(Site_i, max(Clk_{Site_i}))$ and the transitive closure of this operation according to the "happened-before" relation. Figure 5.5 shows the result of extraction the sets of H_{S_i} and H_{max} from the causal history described in figure 5.4.

3. Once H_{S_i} and H_{max} are determined, the group divergence metric can be computed.
For example, we can compute $GD_{tot} = \sum_{i=1}^{3} |H_{max} \setminus H_{S_i}| = 2 + 1 + 3 = 6$

Project name	DVCS	#ChangeSet	#User	#Merge	#Triple	Time (s)
Reddit	git	481	26	6	2444	4
Gollum	git	613	37	41	2851	12
MongoDB	git	13636	91	1992	68186	158
AllTray	Bazaar	389	3	25	2168	5
Anewt	Bazaar	1980	13	45	9433	44
hgview	Mercurial	595	15	32	3257	12
murky	Mercurial	198	17	19	1111	5
anyvc	Mercurial	430	7	4	2172	7

Table 5.2: Execution time and general statistics

1: Let $H_{max} = \{op \in \mathbb{H} : op.timestamp \leq t\}$
2: Let $Sites = \{Site_i : op(Site_i, *) \in H_{max}\}$
3: foreach $Site_i \in Sites$ **do**
4: Let $maxClk = max\{Clk : op(Site_i, Clk) \in H_{max}\}$
5: Let $H_{S_i} = \{op \in H_{max} : op \rightarrow op(Site_i, maxClk)\}$

Listing 5.1: Calculating H_{max} and H_{S_i} for each site starting from a time t

We applied the above algorithm on real software development projects presented in table 5.2. We chose projects managed by different tools (Mercurial, git, Baazar), with different history size (the number of ChangeSets), with different number of users, and different number of merges. The number of merges is an indicator of concurrent activities. We first retrieved the public repositories of these projects to extract H_S for each repository. As these projects extracted from different tools, we transformed their corresponding repositories into causal histories using the SCHO ontology proposed in [7, 5]. We made the following assumptions when parsing the repositories logs:

1. We considered the project as one shared object, so any changeset we find is a modification to this object.

2. We considered each branch in the log as a site.

3. We considered that each site represents one user.

Transformed histories are stored in the JENA semantic triple store [1]. The SCHO ontology allows to implement metrics as SPARQL [2] queries as detailled in [5]. The sixth column of table 5.2 gives the number of triples created by this transformation. The transformation itself is performed by *dvcs2lod*[3] tool; that we developed. This tool handles git, Mercurial and Bazaar repositories.

We implemented a tool called: *DAtool*[4] that slices the history and computes at every step the divergence awareness metric.

Figures 5.6-5.11 show the results obtained after computing group divergence awareness metric on a subset of projects presented in table 5.2.

We sliced the different histories using different time intervals. For small project, we sliced by days, and for big projects, we sliced by months. The

[1] Open source Semantic Web framework for Java: http://openjena.org/
[2] SPARQL Query Language for RDF: http://www.w3.org/TR/rdf-sparql-query/
[3] https://github.com/kmobayed/dvcs2lod
[4] https://github.com/kmobayed/DAtool

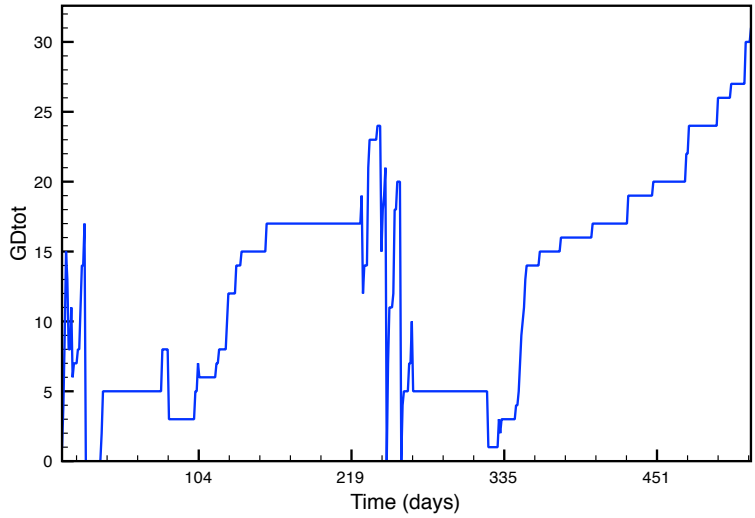


Figure 5.6: GD_{tot} results for Murky project: Mercurial, sliced by day

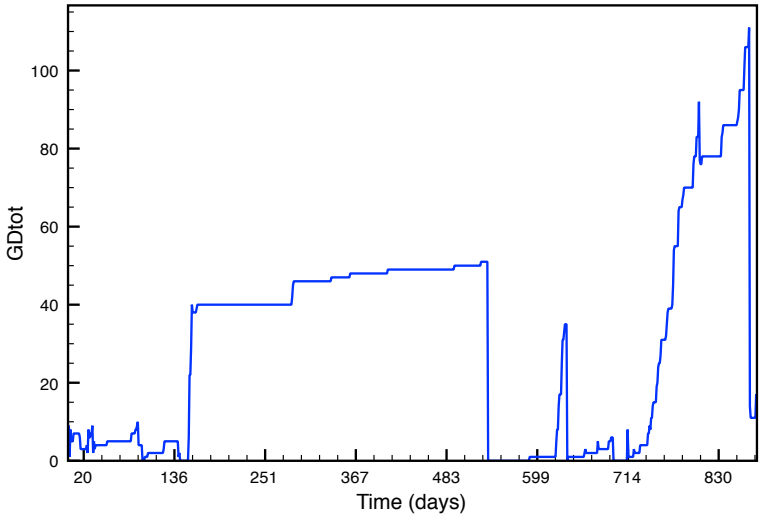


Figure 5.7: GD_{tot} results for hgview project: Mercurial, sliced by day

Y-axis represents the number of *ChangeSets*, while the *X-axis* represents time intervals.

Every figure presents, for a given site, the global divergence (GD_{tot}) at a given time. The graphs display what a user could see during project development if she activated divergence awareness. Displaying such metrics reveal *a posteriori* how much divergence users are tolerating.

For Murky project (see figure 5.6), it is interesting to notice how many times $GD_{tot} = 0$. This means there is no divergence in the system. For this project, it happens only two times on day 2 and day 15 and the maximum divergence concerns 30 ChangeSets.

For Anewt project (see figure 5.8), which is ten times bigger than Murky in term of ChangeSets, divergence average is 100 ChangeSets and maximum of 300 ChangeSets. Convergence is rare, 2 times on analysis time.

For MongoDB project (see figure 5.10), which is ten times bigger than

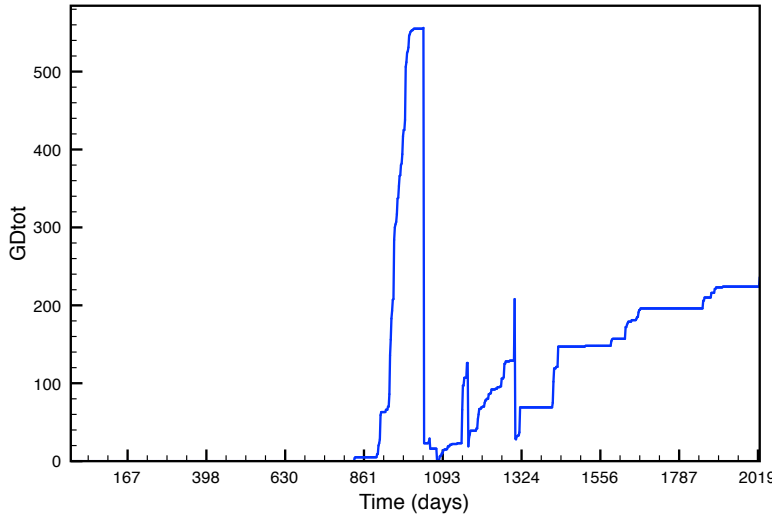


Figure 5.8: GD_{tot} results for Anewt project: Bazaar, sliced by day

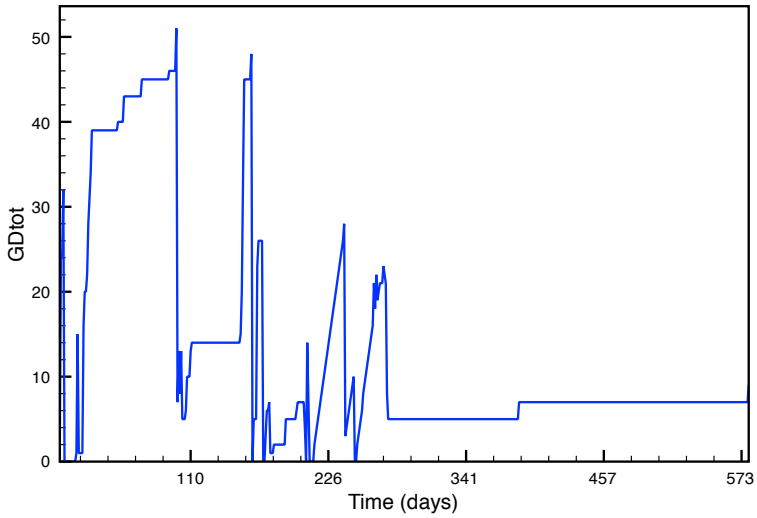


Figure 5.9: GD_{tot} results for allTray project: Bazaar, sliced by day

Anewt in term of ChangeSets, divergence stays between 10 to 30 Changesets and convergence is much more regular. It seems that divergence is managed.

The experimentations demonstrate how group divergence metric can be defined and computed on real histories for analyzing past interactions in multi-synchronous distributed collaborative systems.

5.4 Computing Group Divergence Awareness in Real-Time

In section 5.3, we computed divergence metric on logs i.e. for past interactions. Users can know how much divergence *was* in the system. Divergence awareness has to make users aware about how much divergence *is currently* in the system. This requires to access ongoing unpublished operations for all

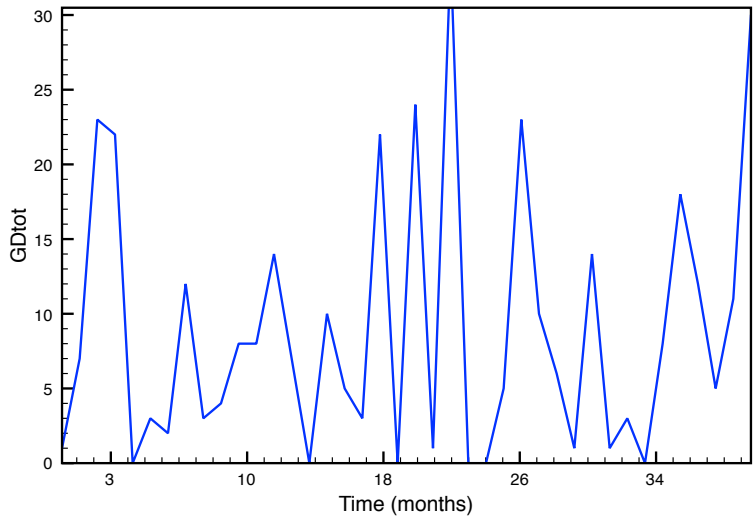


Figure 5.10: GD_{tot} results for mongoDB project: git, sliced by month

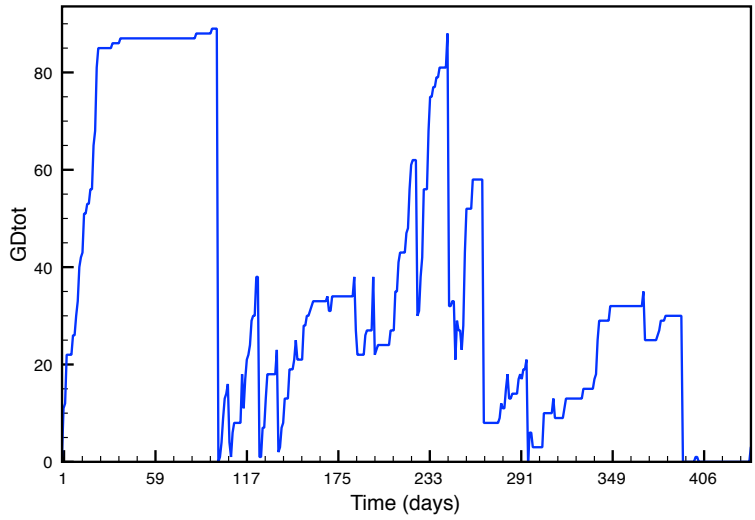


Figure 5.11: GD_{tot} results for gollum project: git, sliced by day

the sites involved in multi-synchronous collaboration. A simple solution for a distributed multi-synchronous collaborative system is to elect a leader and each site publishes and maintains on this central site the number of produced and received operations. The central site computes AGD_{tot} and sends back the results. This approach re-introduces a central site with single point of failure although some distributed multi-synchronous collaborative system such as DVCS or P2P wikis do not require it.

These new distributed multi-synchronous collaborative system can be considered as a social network where each participant follows the updates of the others. A good analogy is to think about a social network such as twitter where messages contain operations that can be executed locally. In such systems, the number and the list of participants are unknown which prevents the computation of AGD_{tot}.

An interesting approach for computing metrics in this context has been

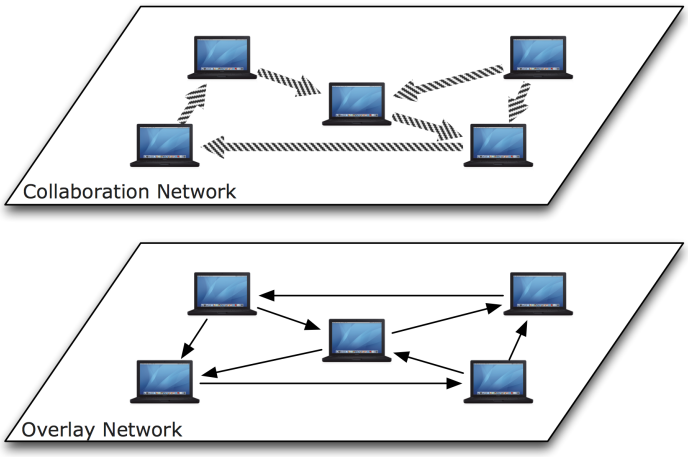


Figure 5.12: Overlay network for exchanging divergence awareness information

developed in [7]. The idea is to discover the topology of the multi-synchronous collaborative system by following the synchronization links between sites. Next, every participant is free to publish some RDF data about its own state, and most important, publish the list of sites it is following. Under these conditions, a distributed semantic query engine such as in [41] can dynamically traverse the multi-synchronous collaborative network, collect informations and compute functions such as AGD_{tot}. This approach can potentially compute AGD_{tot} but it depends on the traversal of the network; failures can stop traversal, it will only work if the network graph is connected.

Another partial approach [52] relies on an overlay network and gossiping algorithms [31]. The idea is to build an overlay network on top of distributed multi-synchronous collaborative system in order to avoid partitioning of the multi-synchronous distributed system (see figure 5.12). In case of failure of one site in the "chain" of followers, the overlay network can repair broken chains and maintain connectivity. This approach does not allow to compute AGD_{tot}, however, it allows to reach every sites in a fully decentralized multi-synchronous collaborative system whatever the basic topology of the social network. The overlay network has probabilistic guarantees to avoid partitioning.

In order to compute AGD_{tot} with the overlay network approach, we will take advantage of the aggregate gossiping algorithm [49]. This family of protocols allows to compute aggregate function on an overlay network without knowing the size of the network. This means that if each site makes available the number of operations it has published and the total number of operations it has received, then AGD_{tot} is computable on the overlay network.

Different algorithms are available for computing aggregate functions with gossiping, we chose the PushSum algorithm [49] to compute AGD_{tot}. PushSum works as follows: At all times t, every node i maintains a sum $sum_{t,i}$, initialized to a local value, and a weight $w_{t,i}$, initialized to $w_{0,i} := 1$ for the node that initiated the protocol while all others start with weight 0. At time 0, it sends the pair $(sum_{0,i}, w_{0,i})$ to itself, and in each subsequent round t, each node i follows the protocol given in listing 5.2. The relative error in the approximation of the sum drops to within ε with probability at least $1-\delta$, in at most $O(logn + log\frac{1}{\varepsilon} + log\frac{1}{\delta})$ rounds.

	$Site_1$			$Site_2$			$Site_3$		
round	$s_{t,1}$	$w_{t,1}$	$op_{tot,1}$	$s_{t,2}$	$w_{t,2}$	$op_{tot,2}$	$s_{t,3}$	$w_{t,3}$	$op_{tot,3}$
0	2.00	1.00	2.00	2.00	0.00	2.00	1.00	0.00	1.00
1	2.50	0.75	3.33	1.00	0.00	1.00	1.50	0.25	6.00
2	1.25	0.37	3.33	2.12	0.31	6.80	1.62	0.31	5.20
3	1.68	0.34	4.90	2.18	0.40	5.38	1.12	0.25	4.50
4	2.35	0.46	5.11	2.07	0.41	5.01	0.56	0.12	4.50
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
8	2.46	0.49	5.00	1.68	0.33	4.99	0.84	0.16	4.98

Table 5.3: A sample execution of the PushSum protocol for calculating op_{tot} for scenario of figure 5.1

```
1: Let {(sum_r, w_r)} be all pairs sent to i in round t − 1
2: Let sum_{t,i} := Σ_r sum_r, w_{t,i} := Σ_r w_r
3: Choose a target f_t(i) uniformly at random
4: Send the pair (½sum_{t,i}, ½w_{t,i}) to f_t(i) and i (yourself)
5: sum_{t,i} / w_{t,i} is the estimate of the average in round t
```

Listing 5.2: PushSum protocol

PushSum protocol can compute an estimation of AGD_{tot} with two PushSum rounds. The first round will calculate the total number of unique operations in the system op_{tot}, by initializing every $sum_{0,i} := Clk_i$. The second round will calculate the total number of operations present on all the sites $\sum_{i=1}^{N} |H_{S_i}|$, by initializing every $sum_{0,i} := |H_{S_i}|$. Then we can directly compute AGD_{tot} using the formula defined in definition 24.

The scenario in figure 5.1 presented a theoretical computation of $op_{tot} = 5$ at step 3. Table 5.3 illustrates the execution of the PushSum protocol for the same scenario. In the initial round, $site_1$ and $site_2$ generated two operations ($sum_{0,1} = 2, sum_{0,2} = 2$), $site_3$ generated one operation ($sum_{0,3} = 1$). We suppose that $site_1$ initiates the algorithm, so $w_{0,1} = 1$, while others are initialized to $w_{0,2} = 0$ and $w_{0,3} = 0$. In every round, the PushSum algorithm is executed. We observe that op_{tot} converges quickly in all sites to the expected value 5. We must notice that in some rounds, the PushSum algorithm can compute value greater than expected value e.g. round 4 on $site_1$, $AGD_{tot} = 5.11$.

This approach computes AGD_{tot} without computing H_{max} on every site, making real-time divergence computation efficient in a fully decentralized deployment of a multi-synchronous distributed collaborative system. This approach does not take into consideration offline sites, since offline sites are unreachable and therefore cannot participate in real-time divergence awareness. Their divergence will be available as soon as they will be online again.

In the next section, we generated different topologies of distributed multi-synchronous collaborative systems and compared the performance of the link traversal and the overlay approach for computing AGD_{tot}.

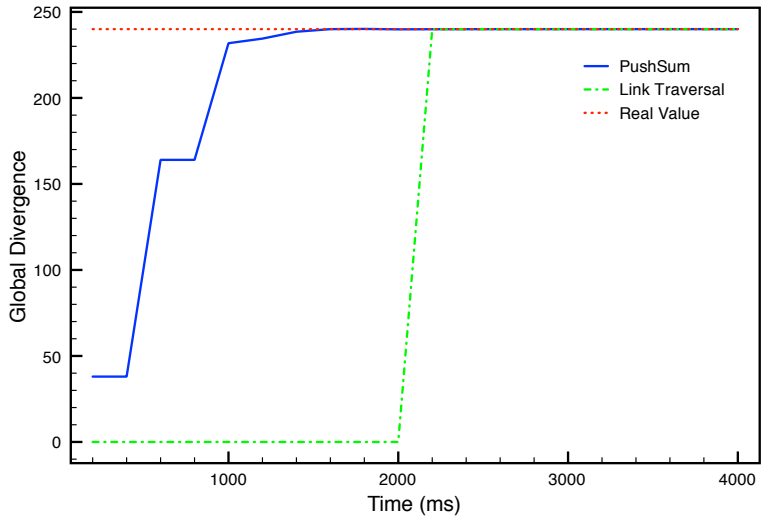


Figure 5.13: GD_{tot} computation time in ms for setup 1: 4 nodes, 2 edges/node

Setup	Network model	#Nodes	#Edges
1	Sample Network	4	8
2	Erdös-Rényi	100	5000
3	Erdös-Rényi	500	125000
4	Erdös-Rényi	1000	10000
5	Barabási-Albert	100	1000
6	Barabási-Albert	500	5000
7	Barabási-Albert	1000	10000

Table 5.4: Multi-synchronous collaboration networks configurations

5.5 Simulating Real-Time Divergence Metrics Computation

We evaluate the performance of PushSum algorithm for real-time divergence metric computation on various settings and compare it with traditional approach based on distributed query evaluation. We deployed the architecture presented in figure 5.12 in the PeerSim simulator [59].

We generate different configurations of multi-synchronous collaboration networks where sites are linked according to different graphs models. We made the hypothesis that there is no isolated sites in the system. We used the Erdös-Rényi [27] random graph model and Barabási-Albert [1] scale-free networks that better represents social graphs. Table 5.4 shows the different characteristics of the generated networks. We generated graphs up to 1000 sites. The third column represents the number of nodes in the network and the last one represents the total number of synchronization links in the system. We used Cytoscape[5] to generate the different graphs.

We compute group divergence AGD_{tot} for every setup in table 5.4 using two methods:

[5]An Open Source Platform for Complex Network Analysis and Visualization:http://www.cytoscape.org/

1. PushSum protocol described in section 5.4.

2. Distributed semantic queries using the Link Traversal Based Query Execution [41]. Link Traversal Based Query Execution approach tries to find relevant data for the query on remote sites by following the synchronization links of every site in the network. AGD_{tot} is represented as a SPARQL query that propagates across the graphs of sites gathering numbers and computes op_{tot} and $\sum_{i=1}^{N} |H_{S_i}|$. This approach is relevant because the topology of sites in the multi-synchronous collaboration network is unknown. Link traversal allows to run a semantic query in these extreme conditions.

For every setup, AGD_{tot} is computed off-line to set the absolute value of group divergence. Next, we can observe how much time in ms is needed by each method to approximate the absolute AGD_{tot} value.

Figure 5.13 shows the results of setup 1: 4 nodes, 2 edges/node. In this network every site is connected to all other sites. Both approaches found the absolute value in less than $3s$. However, the PushSum algorithm approximates the expected value in less than $1.5s$.

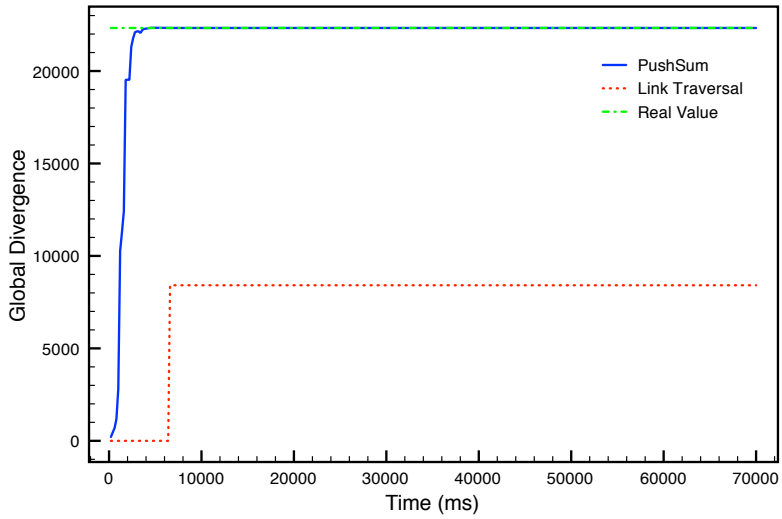


Figure 5.14: AGD_{tot} computation time in ms for setup 2: Erdös-Rényi network, 100 nodes, 50 edges/node

Figure 5.14 shows the results of setup 2: random graph with 100 nodes and 50 edges/node. Both approaches reached a stable value at the same time but the Link Traversal Based Query Execution value is far from the absolute divergence value and it does not succeed in calculating the global divergence. This is because Link Traversal Based Query Execution needs to get the data from all sites, starting from one site, then following the collaboration links in the network. If it cannot reach all sites from this site then result of divergence computation is incorrect. The Link Traversal approach depends on the topology of the multi-synchronous collaboration network, the PushSum approach rebuilds an overlay network on the top of the multi-synchronous collaboration network and consequently does not depend of the internal organization of the

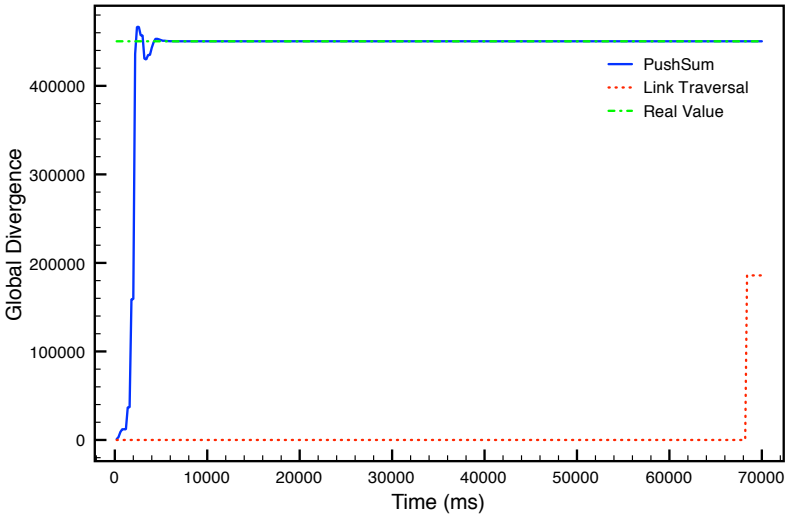


Figure 5.15: AGD_{tot} computation time in ms for setup 3: Erdös-Rényi, 500 nodes, 250 edges/node

multi-synchronous collaboration network. This makes the computation of the divergence metric reliable.

Figure 5.15 shows the results of setup 3: a random graph of 500 nodes with 250 edges/nodes. In this setup, every site is connected to half number of total sites. Here, Link Traversal Based Query Execution is seven times more than the PushSum algorithm to reach the stable value, and even after reaching this value it is still far from the real divergence value. We can also observe that PushSum gets the value quickly but overestimate the value.

Figure 5.16 shows the results of setup 4: a random graph of 1000 nodes with 10 edges/node. As expected, the performance of Link Traversal decreases while PushSum still approximates the good value in $3s$.

Figures 5.17,5.18,5.19 show the results of setup 5,6,7: a scale-free network of 100 to 1000 nodes with 10 edges/nodes. It confirms the previous observations on the relation between the network connectivity and the success of the Link Traversal Based Query. It also confirms that the PushSum algorithm approximates the right value of divergence in less than $4s$ in all experiments.

Simulations demonstrate that it is possible to approximate in few seconds group divergence in large decentralized multi-synchronous collaboration network. The Link Traversal approach is limited by the connectivity of the collaboration network. Maintaining an overlay network independent of the topology of collaboration network and using PushSum algorithm allow to compute the group divergence efficiently. AGD_{tot} can be calculated reliably with the PushSum algorithm while the Link Traversal approach gives no guarantee on the obtained result.

5.6 Related work

In this section, we compare GroupDiv with existing divergence awareness systems described in the section 2.2.2 of the background chapter.

Concurrency awareness [3] is divergence awareness of the past, it detects

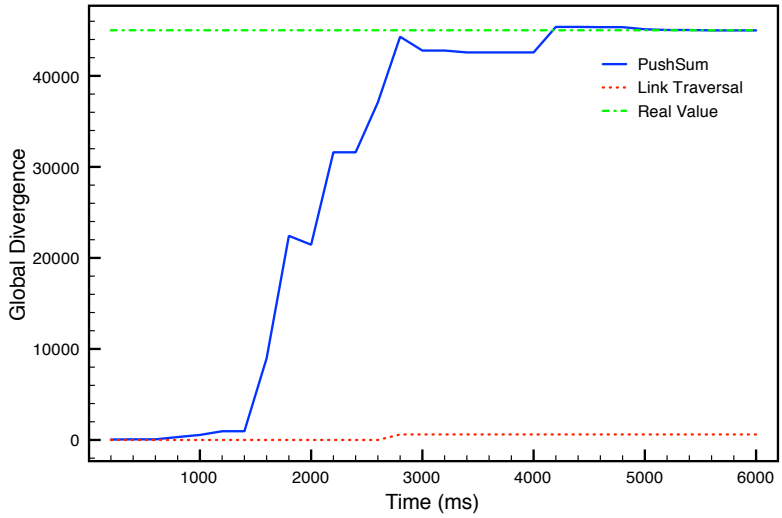


Figure 5.16: AGD_{tot} computation time in ms for setup 4:Erdös-Rényi, 1000 nodes, 10 edges/node

concurrency *a posteriori* to help users to quickly find where automatic merges have been performed in peer-to-peer wikis. If we compare concurrency awareness to group divergence metric, concurrency awareness will be workspace centric i.e. it does not try to compute the number of concurrent operations for all workspaces involved in multi-synchronous collaboration. Even if concurrency awareness can be extended, it relies on plausible clocks that can return false positives.

Crystal [15] is a real-time divergence awareness. It provides developers with concrete information and advice about pending conflicts while remaining largely unobtrusive. Crystal needs access to that developer's repository and the locations of the all other collaborators' repositories. Actually Crystal creates a dedicated repository for awareness computation and integrates all the collaborators' modifications into this repository. Having this centric repository is not compatible with decentralized collaborative systems. Decentralized means that multiple copies of reference can exist. So how to determine a copy of reference. GroupDiv, in some way, determines a virtual copy of reference automatically and measures the distance that leads to this copy.

Concurrency awareness and Crystal divergence awareness systems do not formally define the underlying formula they use to calculate their metrics.

Divergence awareness of the past uses the same metrics as real-time divergence awareness. However, computing real-time metrics is much more challenging. It raises the issue of accessing remote information to compute the metrics in efficient way. It also raises issues related to privacy preservation.

State Treemap [57] is a real-time divergence awareness described in section 2.2.2. In State Treemap the underlying metrics are not precisely defined, there is no formula to compute how much divergence exists in the system. In addition, different users do not see the same treemap. 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 on the workspace the user is working in, it should estimate the entropy relative to global state of the system with all its workspaces.

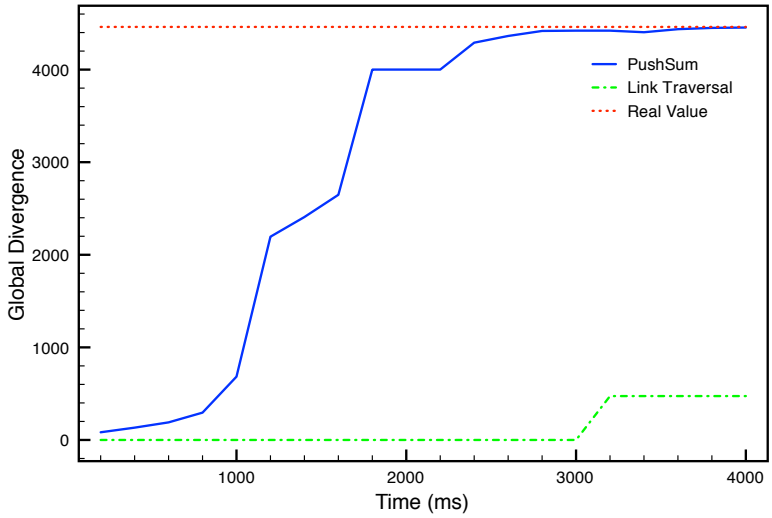


Figure 5.17: AGD_{tot} computation time in ms for setup 5: Barabási-Albert, 100 nodes, 10 edges/node

Divergence awareness is delivered as a treemap where each rectangle is colored with the state of the shared object. State Treemap defines the following divergence awareness states: *Locally Modified:* enables the participant to know that her own copy was modified where the others are not. *Remotely Modified:* makes the participant aware of the changes that occur in the remote workspaces. *Potential Conflict:* means that more than one participant are updating the same document. *Locally Uptodate:* means that there is no new modifications in all remote workspaces.

We can formally declare the previous states for one object using GroupDiv model as follows:

Definition 27 (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 28 (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 29 (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 5.1 (at $step_3$). We will obtain:

State	$Site_1$	$Site_2$	$Site_3$
Locally Modified	true	false	true
Remotely Modified	true	true	true
Potential Conflict	true	false	true
Locally Up-to-date	false	false	false

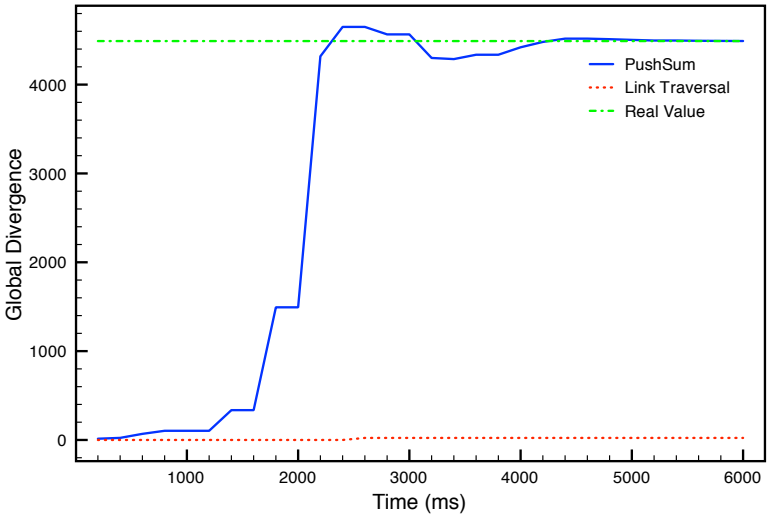


Figure 5.18: AGD_{tot} computation time in ms for setup 6: Barabási-Albert, 500 nodes, 10 edges/node

At the same state, the group divergence metric will give a distance $GD_{tot} = 1 + 3 + 1 = 5$. This clearly demonstrates that State Treemap allows user to perceive divergence and where it is located, but not really to quantify it.

OT divergence awareness [58] is a real-time divergence awareness, in this system sites are synchronized on demand but they exchange unpublished operations in real-time. 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.

Operational transformation relies on the sharing of causal histories as for group divergence metric. One important issue with OT approach is that it should guarantee that all sites will compute the same conflict objects and next will give the same size on each site. It requires to develop quite complex transformation functions and prove convergence properties on them. The group divergence metric defined in this thesis ensures that all sites will see the same value of the metric. In addition, a system can be divergent even with no conflicts (as for $site_2$ in figure 5.1). In this case, a conflict based metric will not capture it. Finally, we think that defining a divergence metric as a distance to next convergence point is more meaningful than the size of conflict to solve.

Ghost operations divergence awareness [46] is a real-time divergence awareness that preserves privacy. Ghost operations are representing real unpublished operations, some parameters of operations can be blurred according to user preferences to better preserve privacy. Ghost operations do not define metrics, they can be considered as an overlay network on top of an existing multi-synchronous collaboration network that deliver ghost operations for each real operation of the system. However, we can observe that computing group divergence metric do not imply sending operations to other sites. The metric computation only requires for each site to publish total number of operations produced (including unpublished ones) and total number of operations received. Sites are not releasing informations on where it is located and what has been done as in ghost operations. Other users can only perceive that an activity exists.

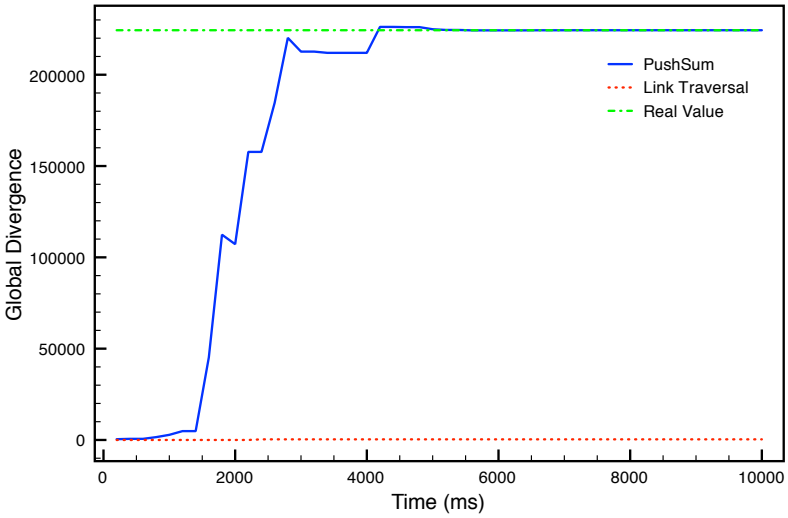


Figure 5.19: AGD_{tot} computation time in ms for setup 7: Barabási-Albert, 1000 nodes, 10 edges/node

Palantir [73] is a divergence awareness tool that provides software developers with insight into others' workspaces. It specifically informs a developer about who changes which artifacts focusing on the concept of conflicts. Conflicts can be direct i.e. concurrent changes on the same artifact or indirect through dependencies between files. Palantir [73] as State Treemap rely 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 with no group membership. Sites just follow the updates of other sites as in social network and generate complex networks of synchronization. In such conditions, it is not possible to deliver divergence awareness that needs to build a global knowledge about the system.

If we interpret Palantir system with GoupDiv defined in section 5.2, sharing events containing operations between all participants in real-time is like building H_{max} in each workspace. Different projections according to various meta-data can be performed locally such as conflict interpretation and impact analysis. However, compared to Palantir, we defined formally a group divergence metric as distance to the next potential convergence state. Palantir does not really define a distance, it tries to estimate size of direct or indirect conflict as in OT divergence awareness [58]. In addition, 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.

In section 3.3, we proposed SCHO [5] an ontology for constructing and sharing the causal history in a distributed collaborative system. SCHO enables defining existing divergence metrics in a declarative way using SPARQL queries. It also makes awareness metrics computation independent of the underlying collaborative system. Different systems can export their histories using SCHO ontology, this makes it possible to compute any existing divergence metric on these logs. In GroupDiv, we used this feature of SCHO to compute group divergence metric for past interactions. As the formal model presented

in section 5.2 is more general than SCHO model, it was possible to express GD_{tot} as a semantic query on the SCHO ontology. More precisely, SCHO represents explicitly the "broadcast" stage of optimistic replication model using $PushFeed$ and $PullFeed$ concepts. Theses feeds represent a publish-subscribe mechanism that allows to ensure causal reception of operations. The GroupDiv model is more general, it makes no difference between published and unpublished operations. Somehow this is important, the metric only depends on local histories and how they were shared. Finally, if we want to apply the SCHO approach to compute metrics in real-time, this means that semantic queries that compute metrics have to be distributed semantic queries. This is what we did in section 5.5 by using the Link traversal approach. The main issue raised by this approach is the discovery of all sites composing the multi-synchronous collaborative system. Link traversal has no guarantee to find all the network, consequently GD_{tot} cannot be safely computed.

5.7 Summary

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 complete existing divergence metrics with a group divergence metric. This original metric is formally defined and can be efficiently computed for past interaction and in real-time fully decentralized systems.

6

Conclusion and Perspectives

Multi-synchronous collaborative systems support parallel stream of activities on replicated data. This enables workspaces to diverge. Divergence allows participants to have different views of shared replicated data. If divergence can help to reduce completion time, it can also generate important overhead through conflicts solving. Divergence awareness is one approach that aims to limit conflicts by making users aware of divergence. It can be seen as an implicit coordination mechanism. Divergence awareness aims to answer the following questions: is there any divergence? With whom? Where? And how much?

Most of existing divergence awareness systems answer the first three questions. Only few ones answer the question related to: How much? Moreover, existing metrics do not estimate a global state of the system with all its workspaces in a fully distributed way.

In this thesis, we focused on quantifying group divergence in decentralized multi-synchronous collaborative systems.

First work on divergence awareness started in 2001 [57]. Many tools, metrics, visualization have been provided later. However, we raised up two main issues about divergence awareness :

1. There is no a common model for understanding, comparing and reasoning on divergence awareness.

2. There is no computation model for computing divergence awareness for the past and in real-time.

In this thesis, we proposed a first model of divergence through the SCHO ontology. In this contribution, we observed that all divergence metrics rely on causality and, next we formalized these causal relations and how this causal graph can be shared between participants using this ontology. We defined algorithms to describe how this ontology is instantiated when users interact following the multi-synchronous collaboration model. This ontology allows a separation of functions between the collaboration tool and the divergence awareness engine. In opposite with existing tools where divergence metrics

are part of the application, the SCHO ontology allows metrics to be written as semantic queries on an abstraction of the interactions. We validated this approach by computing existing divergence metrics on existing traces extracted from different incompatible tools. We also demonstrated how new metrics can be easily defined using the SCHO ontology by computing trust metrics.

However, if the SCHO ontology allows to express divergence metrics as queries, this is just a partial answer for issues concerning common model for understanding and computation of divergence metrics. The SCHO ontology does not provide a clear definition of what is group divergence and it does not define how metrics can be computed in real-time in order to avoid blind modifications.

Next, we pointed out that divergence metrics make sense only within a group and this group has to be defined and maintained through membership management. However, decentralized multi-synchronous collaboration model is clearly an editing social network where each user follows the changes of others. Establishing the membership of divergence metrics requires this social network to be navigable. The SCHO+ ontology is an extension of the SCHO ontology that allows navigation through the editing social network. After that, Link traversal queries can be used both for network discovery and distributed computation of metrics. We simulated network discovery in various setups and demonstrated severe performance issues when the size of the network is growing. If the approach works, it does not scale.

At this point, we proved that divergence metrics can be expressed as queries over an abstraction of a multi-synchronous collaboration model. A decentralized multi-synchronous model can be understood as a editing social network with the "follow your change" relation. We are still lacking of an intuitive definition of group divergence. If it is quite easy to answer is there any divergence ? With who? and Where? The "how much?" question has still unclear definition .

We pointed out that existing metrics are quantifying divergence from a workspace perspective rather than a group perspective i.e. a distance edition from the workspace to a reference copy. We defined an original group divergence metric that measures the distance of the group to the next potential convergence state. The reference copy is now virtual. Each user is able to know her own contribution to this distance. All elements of the metric have been formally defined using an original abstract multi-synchronous collaborative model.

We described how group divergence metric can be computed for past interactions. So participants can know how much divergence *was* in the system. We were able to compute group divergence metric based on real data coming from different distributed version control systems.

We detailed how group divergence metric can be computed in real-time by taking advantage of an overlay network on top of the multi-synchronous collaborative system. The overlay network allows to manage membership issues and to compute aggregate functions. As the group divergence metric can be rewritten with aggregate functions, we obtain an efficient way to compute group divergence metric; each site just maintains the total number of operations it has produced and it has consumed. We built a simulation that compares two approaches for computing group divergence; the first is based

on gossip protocol, the second is based on distributed semantic queries. The simulation demonstrates that aggregate gossip protocols approximate the expected result quickly and reliably. The main problem for distributed semantic queries was related to the membership discovery.

6.1 Perspectives

This work opens several perspectives:

In this thesis, we focused mainly on defining and computing the group divergence metric. In the future, we need to propose widgets to visualize this metric and to conduct usage studies to see how this metric can be really understood and interpreted by users and how users will react to divergence.

Maintaining an overlay network on top of the multi-synchronous collaborative systems opens interesting perspectives. In this thesis, we used it for membership and metrics computations. In the overlay network, each site store only the number of the operations it produced and the number of the operations it consumed. However, it could be also possible also to store references to operations relevant to metric computations, and consequently makes the bridge with existing divergence metrics and on the other hand computing these metrics in a fully distributed system.

In the current proposal, inactive users are handled in the same way as actives ones. However, inactive users will cause group divergence metric to increase till they synchronize their workspaces. It could be interesting to define different type of users and compute the divergence metric with different users projection and compare results. We believe we should detect inactive users to eliminate them from group divergence computation. We can cluster users of the system and determine which users are evolving and group them. The metaphor can be the "tour de france" where distance between groups can be established.

GroupDiv formal model is developed for multi-synchronous collaborative systems. In the future, we want to extend the formal model and make it more suitable for the semantic web world more specially for the Linked Data [42]. We want to develop new divergence awareness metrics for live modification of the linked data. We developed metrics mainly in file synchronization and collaborative text editing. Metrics are not linked to textual data, they are just linked to multi-synchronous collaboration model. In [6, 45], we developed the live linked data approach where we apply multi-synchronous to linked data in semantic web. GroupDiv can help in this context to measure divergence between semantic store replicas and load-balance semantic requests among these replicas. For instance, we are looking to define new metric for Live DBpedia [1]. Once we have these metrics, we can use divergence awareness as an indicator of data freshness in multiple sources or semantic stores during query execution. Divergence awareness metrics will give an indication on the query result accuracy that was executed on a given source.

The missing of the central authority in the decentralized multi-synchronous collaborative system allows to preserve privacy. However, computing divergence awareness requires divulging information that was not originally avail-

[1] http://live.dbpedia.org/

able for the system's users. This can be seen as a privacy breach. Users should be aware of this privacy concern, and work should be done to meet their privacy requirements. For instance, it could be possible to filter the information delivered to divergence awareness engine according to the user privacy rules and it could be possible to use ghost operations as in [46].

SCHO Ontology described in OWL

```
<?xml version="1.0"?>

<!DOCTYPE rdf:RDF [
    <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
    <!ENTITY swrl "http://www.w3.org/2003/11/swrl#" >
    <!ENTITY swrlb "http://www.w3.org/2003/11/swrlb#" >
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY owl2xml "http://www.w3.org/2006/12/owl2-xml#" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
    <!ENTITY scho "http://www.semanticweb.org/ontologies/2009/4/scho.owl#" >
]>


<rdf:RDF xmlns="http://www.semanticweb.org/ontologies/2009/4/scho.owl#"
     xml:base="http://www.semanticweb.org/ontologies/2009/4/scho.owl"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:swrl="http://www.w3.org/2003/11/swrl#"
     xmlns:owl2xml="http://www.w3.org/2006/12/owl2-xml#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:scho="http://www.semanticweb.org/ontologies/2009/4/scho.owl#">
    <owl:Ontology rdf:about=""/>


<!--
    ///////////////////////////////////////////////
    //
    // Classes
    //
    ///////////////////////////////////////////////
     -->

    <owl:Class rdf:about="&owl;Thing"/>
```

```
<owl:Class rdf:about="#Site">
    <rdfs:subClassOf rdf:resource="&owl;Thing"/>
</owl:Class>

<owl:Class rdf:about="#Document">
    <rdfs:subClassOf rdf:resource="&owl;Thing"/>
</owl:Class>

<owl:Class rdf:about="#ChangeSet">
    <rdfs:subClassOf rdf:resource="&owl;Thing"/>
</owl:Class>

<owl:Class rdf:about="#Patch">
    <rdfs:subClassOf rdf:resource="&owl;Thing"/>
</owl:Class>

<owl:Class rdf:about="#Operation">
    <rdfs:subClassOf rdf:resource="&owl;Thing"/>
</owl:Class>

<owl:Class rdf:about="#PullFeed">
    <rdfs:subClassOf rdf:resource="&owl;Thing"/>
</owl:Class>

<owl:Class rdf:about="#PushFeed">
    <rdfs:subClassOf rdf:resource="&owl;Thing"/>
</owl:Class>

<!--
///////////////////////////////////////////////
//
// Object Properties
//
///////////////////////////////////////////////
 -->


<owl:ObjectProperty rdf:about="#hasOp">
    <rdfs:domain rdf:resource="#Patch"/>
    <rdfs:range rdf:resource="#Operation"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#hasDoc">
    <rdfs:domain rdf:resource="#Site"/>
    <rdfs:range rdf:resource="#Document"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#hasPatch">
    <rdfs:domain rdf:resource="#ChangeSet"/>
    <rdfs:range rdf:resource="#Patch"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#hasPull">
    <rdfs:domain rdf:resource="#Site"/>
    <rdfs:range rdf:resource="#PullFeed"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#hasPullHead">
```

```
    <rdfs:domain rdf:resource="#PullFeed"/>
    <rdfs:range rdf:resource="#ChangeSet"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#hasPush">
    <rdfs:domain rdf:resource="#Site"/>
    <rdfs:range rdf:resource="#PushFeed"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#hasPushHead">
    <rdfs:domain rdf:resource="#PushFeed"/>
    <rdfs:range rdf:resource="#ChangeSet"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#head">
    <rdfs:domain rdf:resource="#Document"/>
    <rdfs:range rdf:resource="#Patch"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#inPullFeed">
    <rdfs:domain rdf:resource="#ChangeSet"/>
    <rdfs:range rdf:resource="#PullFeed"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#inPushFeed">
    <rdfs:domain rdf:resource="#ChangeSet"/>
    <rdfs:range rdf:resource="#PushFeed"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#onDoc">
    <rdfs:domain rdf:resource="#Patch"/>
    <rdfs:range rdf:resource="#Document"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#previous">
    <rdfs:domain rdf:resource="#Patch"/>
    <rdfs:range rdf:resource="#Patch"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#previousChangeSet">
    <rdfs:domain rdf:resource="#ChangeSet"/>
    <rdfs:range rdf:resource="#ChangeSet"/>
</owl:ObjectProperty>


</rdf:RDF>
```

Bibliography

[1] R. Albert and A.L. Barabasi. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002. 78, 96

[2] Larry Allen, Gary Fernandez, Kenneth Kane, David B. Leblang, Debra Minard, and John Posner. ClearCase MultiSite: Supporting Geographically-Distributed Software Development. In *Software Configuration Management, SCM'95*, pages 194–214. Springer, 1995. 7, 19, 20, 49, 62, 69, 71

[3] Sawsan Alshattnawi, Gérôme Canals, and Pascal Molli. Concurrency awareness in a P2P wiki system. In *International Symposium on Collaborative Technologies and Systems*, pages 285–294. IEEE, May 2008. 3, 8, 33, 34, 43, 82, 98

[4] L. A. N. Amaral, A. Scala, M. Barthélémy, and H. E. Stanley. Classes of small-world networks. *Proceedings of the National Academy of Sciences*, 97(21):11149–11152, October 2000. 67

[5] Khaled Aslan, Nagham Alhadad, Hala Skaf-Molli, and Pascal Molli. SCHO: An Ontology Based Model for Computing Divergence Awareness in Distributed Collaborative Systems. In *The Twelfth European Conference on Computer-Supported Cooperative Work, ECSCW2011*, pages 373–392. Springer, September 2011. 49, 72, 88, 90, 102

[6] Khaled Aslan, Pascal Molli, and Hala Skaf-Molli. C-Set: a Commutative Replicated Data Type for Semantic Stores. In *Fourth International Workshop on Resource Discovery*, Heraklion, Greece, 2011. 107

[7] Khaled Aslan, Hala Skaf-molli, and Pascal Molli. 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. 82, 88, 90, 94

[8] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The description logic handbook: theory, implementation, and applications*. Cambridge University Press, New York, NY, USA, 2003. 47

[9] Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. Gephi: An Open Source Software for Exploring and Manipulating Networks. In *International AAAI Conference on Weblogs and Social Media*. AAAI, 2009. 67

[10] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, 284(5):34–43, May 2001. 44

[11] Peter Biddle, Paul Engl, Marcus Peinado, and Bryan Willman. The darknet and the future of content distribution. In *In Proceedings of the 2002 ACM Workshop on Digital Rights Management*, 2002. 63

[12] Jacob T. Biehl, Mary Czerwinski, Greg Smith, and George G. Robertson. Fastdash: a visual dashboard for fostering awareness in software teams. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1313–1322. ACM, 2007. 3, 40

[13] Dan Brickley and Ramanathan V. Guha. Rdf vocabulary description language 1.0: Rdf schema. http://www.w3.org/TR/rdf-schema/, 2004. 45

[14] Dan Brickley and Libby Miller. Foaf: The friend of a friend project. http://www.foaf-project.org/. 65, 70

[15] Yuriy Brun, Reid Holmes, Michael D. Ernst, and David Notkin. 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)*, pages 168–178, 2011. 3, 8, 9, 28, 34, 35, 82, 99

[16] Reidar Conradi and Bernhard Westfechtel. Version models for software configuration management. *ACM Computer Survey*, 30(2):232–282, June 1998. 19

[17] C.R.B. de Souza, D. Redmiles, and P. Dourish. 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*, pages 105–114. ACM, 2003. 8, 81

[18] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic Algorithms for Replicated Database Maintenance. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 1–12, Vancouver, British Columbia, Canada, August 1987. ACM Press. 22

[19] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, PODC '87, pages 1–12. ACM, 1987. 72

[20] Prasun Dewan and R. Hegde. Semi-synchronous conflict detection and resolution in asynchronous software development. In *The Sixth European Conference on Computer-Supported Cooperative Work, ECSCW2007*, pages 159–178. Springer, 2007. 8, 28

[21] A. Dix, J. Finlay, G. Abowd, and R. Beale. *Human-Computer Interaction*. Prentice Hall, 1997. 3, 28, 29, 30

[22] Alan Dix. Challenges for cooperative work on the web: An analytical approach. *Computer Supported Cooperative Work (CSCW)*, 6:135–156, 1997. 3, 28, 29, 30

[23] Paul Dourish. The Parting of the Ways: Divergence, Data Management and Collaborative Work. In *4th European Conference on Computer Supported Cooperative Work*, pages 215–230. Kluwer Academic Publishers, 1995. 7, 18, 83

[24] Paul Dourish and Victoria Bellotti. Awareness and coordination in shared workspaces. In *Proceedings of ACM conference on Computer-Supported Cooperative Work*, pages 107–114. ACM Press, 1992. 29, 30

[25] Edd Dumbill. Doap: Description of a project. http://trac.usefulinc.com/doap. 70

[26] Clarence A. Ellis, Simon J. Gibbs, and Gail Rein. Groupware: some issues and experiences. *Communication of the ACM*, 34(1):39–58, January 1991. 15, 17

[27] P. Erdos and A. Rényi. *On the evolution of random graphs*. Akad. Kiadó, 1960. 78, 96

[28] Jacky Estublier and Sergio Garcia. Process model and awareness in SCM. In *Proceedings of the 12th international workshop on Software configuration management*, volume 12, pages 59–74. ACM, 2005. 8, 28

[29] Patrick T. Eugster, Rachid Guerraoui, Sidath B. Handurukande, Petr Kouznetsov, and Anne-Marie Kermarrec. Lightweight Probabilistic Broadcast. *ACM Transactions on Computer Systems*, 21(4):341–374, November 2003. 22

[30] L Freeman. Centrality in social networks conceptual clarification. *Social Networks*, 1(3):215–239, 1979. 67, 71

[31] Ayalvadi J. Ganesh, Anne-Marie Kermarrec, and Laurent Massoulié. Peer-to-peer membership management for gossip-based protocols. *IEEE Trans. Comput.*, 52(2):139–149, February 2003. 94

[32] Saul Greenberg, Carl Gutwin, and Mark Roseman. Semantic Telepointers for Groupware. In *Proceedings of the 6th Australian Conference on Computer-Human Interaction*, page 54. IEEE Computer Society, 1996. 31

[33] Ralph Gross, Alessandro Acquisti, and H. John Heinz, III. Information revelation and privacy in online social networks. In *WPES '05: Proceedings of the 2005 ACM workshop on Privacy in the electronic society*, pages 71–80, 2005. 62, 76

[34] Tom Gross, Chris Stary, and Alex Totter. User-centered awareness in computer-supported cooperative work-systems: Structured embedding of findings from social sciences. *International Journal of Human-Computer Interaction*, 18(3):323–360, July 2005. 8, 82

[35] W3C OWL Working Group. Owl 2 web ontology language. http://www.w3.org/TR/owl2-overview/, 2009. 45

[36] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, June 1993. 47, 48

[37] N. Guarino and P. Giaretta. Ontologies and knowledge bases: Towards a terminological clarification. In N. Mars, editor, *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing*, pages 25–32. IOS Press, 1995. 47

[38] Nicola Guarino, Daniel Oberle, and Steffen Staab. What is an ontology? In Steffen Staab and Rudi Studer, editors, *Handbook on Ontologies*, International Handbooks on Information Systems, pages 1–17. Springer Berlin Heidelberg, 2009. 47

[39] Carl Gutwin and Saul Greenberg. Effects of awareness support on groupware usability. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 511–518. ACM Press/Addison-Wesley Publishing Co., 1998. 8, 82

[40] Carl Gutwin and Saul Greenberg. A descriptive framework of workspace awareness for real-time groupware. *Computer Supported Cooperative Work*, 11(3):411–446, November 2002. 31

[41] Olaf Hartig, Christian Bizer, and Johann Christoph Freytag. Executing SPARQL Queries over the Web of Linked Data. In *International Semantic Web Conference*, pages 293–309. Springer Berlin, Heidelberg, 2009. 70, 77, 78, 94, 97

[42] Tom Heath and Christian Bizer. *Linked Data: Evolving the Web into a Global Data Space*. Synthesis Lectures on the Semantic Web. Morgan & Claypool Publishers, 2011. 107

[43] Martin Hepp, Pieter De Leenheer, and Aldo de Moor. Ontologies: State of the Art, Business Potential, and Grand Challenges. *Ontology Management: Semantic Web, Semantic Web Services, and Business Applications*, pages 3–22, 2007. 47

[44] Jason Hill. The MAUI toolkit: Groupware widgets for group awareness. *Computer Supported Cooperative Work (CSCW)*, 13(5-6):539–571, 2004. 30

[45] Luis Daniel Ibáñez, Hala Skaf-Molli, Pascal Molli, and Olivier Corby. Synchronizing semantic stores with commutative replicated data types. In Alain Mille, Fabien L. Gandon, Jacques Misselis, Michael Rabinovich, and Steffen Staab, editors, *WWW (Companion Volume)*, pages 1091–1096. ACM, 2012. 107

[46] Claudia-Lavinia Ignat, Gérald Oster, Pascal Molli, and Hala Skaf-Molli. A Collaborative Writing Mode for Avoiding Blind Modifications. In *Ninth International Workshop on Collaborative Editing Systems, GROUP'07*, pages 1–6, 2007. 3, 8, 26, 32, 38, 81, 82, 101, 108

[47] Robert Johansen. *Groupware: Computer Support for Business Teams*. The Free Press, 1988. 7, 15

[48] P. Johnson and R. Thomas. RFC677: The maintenance of duplicate databases. 1976. 72

[49] David Kempe, Alin Dobra, and Johannes Gehrke. Gossip-based computation of aggregate information. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '03, pages 482–491, Washington, DC, USA, 2003. IEEE Computer Society. 94

[50] Michael Kifer, Georg Lausen, and James Wu. Logical foundations of object-oriented and frame-based languages. *J. ACM*, 42(4):741–843, July 1995. 48

[51] Leslie Lamport. Times, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7):558–565, 1978. 22, 23, 48, 71, 83, 84

[52] E. Le Merrer and Gilles Straub. Distributed Overlay Maintenance with Application to Data Consistency. In *Globe 2011, 4th International Conference on Data Management in Grid and P2P Systems*, pages 25–36, 2011. 94

[53] Frank Manola and Eric Miller. Rdf primer. http://www.w3.org/TR/rdf-primer/, 2004. 45

[54] Friedemann Mattern. Virtual time and global states of distributed systems. In *International Workshop on Parallel and Distributed Algorithms*, pages 215–226, 1989. 72, 84

[55] Microsoft. Microsoft office groove. http://office.microsoft.com/en-us/groove/, 2005. 63

[56] Pascal Molli, Gérald Oster, Hala Skaf-Molli, and Abdessamad Imine. Using the transformational approach to build a safe and generic data synchronizer. In Kjeld Schmidt, Mark Pendergast, Marilyn Tremaine, and Carla Simone, editors, *GROUP*, pages 212–220. ACM, 2003. 24

[57] Pascal Molli, Hala Skaf-Molli, and Christophe Bouthier. State Treemap: an awareness widget for multi-synchronous groupware. In *Seventh International Workshop on Groupware - CRIWG*, pages 106–114. IEEE Computer Society, 2001. 3, 8, 9, 20, 36, 43, 82, 99, 105

[58] Pascal Molli, Hala Skaf-Molli, and Gérald Oster. Divergence awareness for virtual team through the web. In *Integrated Design and Process Technology, IDPT 2002*, pages 1–10, Pasadena, CA, USA, June 2002. Society for Design and Process Science. 3, 8, 9, 28, 30, 32, 37, 40, 81, 82, 101, 102

[59] Alberto Montresor and Mark Jelasity. PeerSim: A scalable P2P simulator. In *2009 IEEE Ninth International Conference on Peer-to-Peer Computing*, pages 99–100. Ieee, September 2009. 96

[60] Sean A. Munson, Emily Rosengren, and Paul Resnick. Thanks and tweets: comparing two public displays. In Pamela J. Hinds, John C. Tang, Jian Wang, Jakob E. Bardram, and Nicolas Ducheneaut, editors, *CSCW '11: Proceedings of the ACM 2011 conference on Computer supported cooperative work*, pages 331–340. ACM, 2011. 17

[61] Natalya F. Noy and Deborah L. McGuinness. Ontology Development 101: A Guide to Creating Your First Ontology. Stanford Knowledge Systems Laboratory Technical Report KSL-01-05, 2001. 47

[62] Gérald Oster, Pascal Urso, Pascal Molli, and Abdessamad Imine. Data Consistency for P2P Collaborative Editing. In *Conference on Computer-Supported Cooperative Work*, 2006. 23

[63] Stavroula Papadopoulou, Claudia Ignat, Gerald Oster, and Moira Norrie. Increasing Awareness in Collaborative Authoring through Edit Profiling. In *IEEE Conference on Collaborative Computing: Networking, Applications and Worksharing - CollaborateCom 2006*, pages 1–9, 11 2006. 3, 8, 33, 43, 82

[64] D.E. Perry, H.P. Siy, and L.G. Votta. Parallel changes in large-scale software development: an observational case study. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 10(3):308–337, 2001. 8, 81

[65] Nuno Preguica, Joan Manuel Marques, Marc Shapiro, and Mihai Letia. A Commutative Replicated Data Type for Cooperative Editing. In *The 29th IEEE International Conference on Distributed Computing Systems*, pages 395–403. IEEE, June 2009. 53

[66] João Gustavo Prudêncio, Leonardo Murta, Cláudia Werner, and Rafael Cepêda. To lock, or not to lock: That is the question. *Journal of Systems and Software*, 85(2):277–289, February 2012. 8, 81

[67] Charbel Rahhal, Hala Skaf-Molli, Pascal Molli, and Stéphane Weiss. Multi-synchronous Collaborative Semantic Wikis. In *10th International Conference on Web Information Systems Engineering - WISE '09*, volume 5802 of *LNCS*, pages 115–129. Springer, October 2009. 9, 44, 48, 51, 63, 82

[68] Michel Raynal. About logical clocks for distributed systems. *SIGOPS Operating Systems Review*, 26(1):41–48, January 1992. 84

[69] Michael Rogers and Saleem Bhatti. How to disappear completely: A survey of private peer-to-peer networks. In *Sustaining Privacy in Autonomous Collaborative Encironments (SPACE 2007)*, 2007. 63

[70] Mark Roseman and Saul Greenberg. Building real-time groupware with groupkit, a groupware toolkit. *ACM Transactions on Computer-Human Interaction*, 3(1):66–106, March 1996. 17

[71] Yasushi Saito and Marc Shapiro. Optimistic replication. *ACM Computing Surveys*, 37(1):42–81, March 2005. 22, 48, 70, 83

[72] Anita Sarma, Zahra Noroozi, and André van der Hoek. Palantir: raising awareness among configuration management workspaces. In *Proceedings of the 25th International Conference on Software Engineering*, ICSE '03, pages 444–454. IEEE Computer Society, 2003. 43

[73] Anita Sarma, David Redmiles, and André Van der Hoek. Palantir: Early Detection of Development Conflicts Arising from Parallel Code Changes. *IEEE Transactions on Software Engineering*, 99, 2011. 3, 8, 38, 39, 82, 102

[74] Evan Schoenberg and Zachary West. Adium : Instant messaging client. http://www.adium.im/, 2001. 66

[75] N. Shadbolt, Tim Berners-Lee, and W. Hall. The Semantic Web Revisited. *IEEE Intelligent Systems*, 21(3):96–101, May 2006. 44

[76] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. Conflict-free Replicated Data Types. In *The 13th International Symposium on Stabilization, Safety, and Security of Distributed Systems*, pages 386–400. Springer, 2011. 53

[77] Skaf and et al. Dsmw : Distributed semantic media wiki. http://momo54.github.com/DSMW/, 2009. 8, 19

[78] Hala Skaf-Molli, Claudia-Lavinia Ignat, Charbel Rahhal, and Pascal Molli. New Work Modes for Collaborative Writing. In *Enterprise Information Systems and Web Technologies*, pages 176–182, 2007. 7, 18

[79] Hala Skaf-Molli, Charbel Rahhal, and Pascal Molli. Peer-to-peer semantic wikis. In *20th International Conference on Database and Expert Systems Applications- DEXA 2009, Lecture Notes in Computer Science 5690, Springer*, volume 5690 of *Lecture Notes in Computer Science*, Linz, Austria, August 2009. Springer. 23

[80] Jason Stewart, Benjamin B. Bederson, and Allison Druin. Single display groupware: a model for co-present collaboration. In *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit*, CHI '99, pages 286–293, New York, NY, USA, 1999. ACM. 17

[81] Chengzheng Sun, X. Jia, Y. Zhang, Y. Yang, and D. Chen. Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 5(1):63–108, 1998. 22, 24, 37, 48, 71, 83, 84

[82] J. Tam and S. Greenberg. A framework for asynchronous change awareness in collaborative documents and workspaces. *International Journal of Human-Computer Studies*, 64(7):583–598, 2006. 30, 31

[83] Francisco J. Torres-Rojas and Mustaque Ahamad. Plausible clocks: constant size logical clocks for distributed systems. *Distributed Computing*, 12:179–195, 1999. 33

[84] Wil van der Aalst and Kees van Hee. *Workflow Management: Models, Methods, and Systems (Cooperative Information Systems)*. The MIT Press, 2002. 18

[85] Stéphane Weiss, Pascal Urso, and Pascal Molli. Wooki: a P2P Wiki-based Collaborative Writing Tool. In *Proceedings of the 8th International Conference on Web Information Systems Engineering*, pages 503–512. Springer, 2007. 23, 33

[86] Stéphane Weiss, Pascal Urso, and Pascal Molli. Logoot : a Scalable Optimistic Replication Algorithm for Collaborative Editing on P2P Networks. In *International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2009. 23

[87] Stéphane Weiss, Pascal Urso, and Pascal Molli. Logoot-Undo: Distributed Collaborative Editing System on P2P Networks. *IEEE Transactions on Parallel and Distributed Systems*, 21(8):1162 – 1174, 2010. 53

[88] Jan Wloka, Barbara Ryder, Frank Tip, and Xiaoxia Ren. Safe-commit analysis to facilitate team software development. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE '09, pages 507–517, Washington, DC, USA, 2009. IEEE Computer Society. 28

[89] Thomas Zimmermann. Mining Workspace Updates in CVS. In *Proceedings of the Fourth International Workshop on Mining Software Repositories, ICSE Workshops MSR'07*, pages 11–11, May 2007. 28

l'université nantes angers le mans
PÔLE DE RECHERCHE ET D'ENSEIGNEMENT SUPÉRIEUR

UNIVERSITÉ DE NANTES

Thèse de Doctorat

Khaled Aslan Almoubayed

Divergence Awareness in Distributed Multi-Synchronous Collaborative Systems

Résumé

Les systèmes collaboratifs peuvent être synchrones, asynchrones, ou multi-synchrones. Dans les systèmes collaboratifs multi-synchrones, les participants travaillent en parallèle sur des copies locales d'objets partagés. Ils synchronisent leurs modifications de temps-en-temps pour assurer un état cohérent. Le modèle de collaboration multi-synchrone introduit la notion de divergence entre copies d'objets partagés. Travailler en parallèle peut potentiellement réduire le temps de réalisation des tâches. Cependant, il introduit des modifications à l'aveugle et le coût de résolution des conflits introduits par les modifications concurrentes peut surpasser le gain attendu. *Divergence awareness* quantifie la divergence et répond aux questions suivantes : y a-t-il divergence ? avec qui ? où ? et combien ? Les mesures existantes quantifient la divergence du point de vue de l'utilisateur et non du groupe. Je vais adresser le problème de divergence de groupe qui répond spécifiquement à la question "combien ?". Cela permet aux utilisateurs de devenir conscients de la distance du groupe au prochain point de convergence potentiel.
Ce travail présente un modèle générique pour définir une abstraction des systèmes multi-synchrones. Et propose une métrique de divergence de groupe et montre comment les métriques existantes peuvent être exprimées dans ce modèle. Il propose également un algorithme efficace pour calculer la métrique de divergence de groupe dans un réseau entièrement décentralisé.

Mots clés

systèmes distribués, systèmes collaboratifs, web sémantique, awareness.

Abstract

Multi-synchronous collaborative systems support parallel streams of activities on replicated data. They allow streams of activities to diverge. If divergence can help to reduce completion time, it can also generate important overhead when solving conflicts. Divergence awareness is one approach that aims to limit conflicts by making users aware of divergence. It aims to answer the following questions: is there any divergence? With whom? Where? And how much?
Existing divergence awareness metrics are highly coupled to their original applications and can not be used outside their original scope. In addition, existing divergence awareness do not estimate a global state of the system with all its workspace in a fully distributed way.
In this thesis, I propose a formal model to express existing divergence awareness metrics. I propose also an original group divergence metric that addresses specifically the "how much?" question. This metric makes users aware of the distance of the group to the next potential convergence point. I define formally the group divergence awareness metric. Next, I propose an algorithm to compute group divergence metric on logs and validates the algorithm with real data from different development projects. Finally, I propose an original approach based on overlay network to compute group divergence metric in real-time in a fully decentralized network and validate the approach with simulations.

Key Words

Multi-synchronous Collaboration, Divergence Awareness, Distributed System, Semantic Web.