SCHO: An Ontology Based Model for Computing Divergence Awareness in Distributed Collaborative Systems

Khaled Aslan, Nagham Alhadad, Hala Skaf-Molli, Pascal Molli

SCHO: An Ontology Based Model for Computing Divergence Awareness in Distributed Collaborative Systems

Khaled Aslan, Nagham Alhadad, Hala Skaf-Molli, Pascal Molli
LINA, Université de Nantes, France
khaled.aslan-almoubayed@univ-nantes.fr, nagham.alhadad@univ-nantes.fr, hala.skaf@univ-nantes.fr, pascal.molli@univ-nantes.fr

Abstract. 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.

Introduction

Web 2.0 applications showed the importance of collaborative systems. These systems changed the internet users into web writers. Many collaboration tools are currently available like: Wikis, blogs, video-conferencing systems, version control systems. These collaboration tools are classified using Ellis matrix (Ellis and Gibbs, 1989). This matrix is based on two dimensions; time and space. Users can be distributed in time and space. But there is another dimension which is working on different copies. This collaboration mode is called multi-synchronous (Dourish, 1995). In this mode users replicate shared objects and they converge or diverge

among each other. This divergence can be observed, measured and visualized. This is called divergence awareness.

Many previous work have addressed the measurement and the visualization of the divergence. Divergence awareness is provided in different systems with ad-hoc visualizations, such as State Treemap (Molli et al., 2001), Palantir (Sarma et al., 2003), Edit Profile (Papadopoulou et al., 2006), Wooki (Weiss et al., 2007), etc.

Existing systems define their own divergence metrics without a common formal definition. Metrics are coupled with the application and cannot be used outside their original scope. There is no previous work that tried to build a unified formal model for divergence awareness. Unified model for computing divergence opens the opportunity to build a middleware for distributed collaborative systems.

Multi-synchronous collaborative systems rely on optimistic replication models (Rahhal et al., 2009). Optimistic replication models are based on history sharing which are application independent. We propose to conceptualize and formalize the sharing of causal history in distributed collaborative system. This allows to compute divergence metrics in a declarative way independently of the application.

In this paper, we use semantic web technology 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 based on this model. We validate our work using real data extracted from software engineering development projects.

The paper is structured as follows. First section presents divergence metrics in distributed collaborative systems. The second section details the SCHO ontology. The third section presents the formal definition of divergence awareness metrics based on this ontology. The fourth section presents the validation of the proposed model using real data. The last section concludes the paper.

Divergence Metrics in Distributed Collaborative Systems

Divergence occurs when there are more than one copy of a shared document and participants can modify their copies in parallel. A cooperative work is a cycle of divergence and convergence. Divergence occurs when two activities have different views of a shared document. After divergence participants synchronize their activities to reestablish a common view of the document; further individual activities will cause divergence again, necessitating further synchronization and so on (Dourish, 1995). State Treemap (Molli et al., 2001), Palantir (Sarma et al., 2003), Edit Profile (Papadopoulou et al., 2006), Wooki (Weiss et al., 2007) are examples of tools that provide the user with divergence awareness.

State Treemap

State Treemap(Molli et al., 2001) is a divergence awareness mechanism, it enables the participant to be aware of the differences among her documents and the others' documents by using the different document 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.
- *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 has been modified by a participant, it will be marked as (Local Modified) in her own workspace, where in the others participants' workspaces it will be marked as (Remotely Modified).

Palantir

Palantir (Sarma et al., 2003) is another awareness tool that provides software developers with insight into others' workspaces. It gives awareness information about concurrent modifications performed in isolation in the context of configuration management systems. Palantir captures a number of events, for instance, *added, removed, changes in progress, changes committed,* etc. The effect of the changes is computed and presented to the users by means of two metrics: the severity and impact metrics. The severity metric measures how much a component in a user's workspace has changed when compared to its latest checked version in the repository, or its version on the collaborators workspaces. It can be binary, indicating that a change of any kind occurred, or a number indicating the percentage of lines of code that were modified in any way. The impact measure takes into consideration the code dependencies and computes how much a component in a local workspace is affected by changes to related components in remote workspaces. Table I summarizes the awareness states defined in Palantir proposal.

Edit Profile

Edit Profile (Papadopoulou et al., 2006) provides awareness 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.

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 has changed

Table I. Palantir divergence awareness states.

Wooki (Concurrent Modifications)

Wooki (Weiss et al., 2007) is a P2P wiki composed of a P2P network of autonomous wiki servers. Each wiki page is replicated over the whole set of server. A change performed on a wiki server is immediately applied to the local copy and then propagated to the other sites. A remote change when received by a server is merged with local changes and then applied to the local copy. To avoid the conflicts, the redundancy or even the unreliable information that result from the merge, a concurrency awareness mechanism is used. This mechanism makes users aware of the status of the pages they access regarding concurrency: Has this page been merged automatically? In this case show me where concurrent changes occur since the last reviewed state?

Existing systems such as State Treemap, Palantir, Edit Profile and Wooki; define their own divergence metrics without a common formal definition. Metrics are coupled with the application and cannot be used outside their original scope. In the next section, we propose a unified formal model for divergence awareness. This model allows to compute all existing divergence metrics.

SCHO: Shared Causal History Ontology

Divergence occurs when there are more than one copy of a shared document. Optimistic replication model (Saito and Shapiro, 2005) 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 (Sun et al., 1998). Causality ensures that all operations are ordered by a precedence relation in the sense of the Lamport's happened-before relation (Lamport, 1978). Therefore they will be executed in same order on every site. Broadcast-

ing 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 (Rahhal et al., 2009). The model is not application dependent. Consequently, we have to determine the underlying concepts required to exchange change sets 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.
- 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.

Unified Shared Causal History Model (SCHO)

The Shared Causal History Ontology (SCHO) shown in Figure 1 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. This ontology is populated through the user interaction 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 (Allen et al., 1995). 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
- createPush: Creates a topic (or feed) in the publish/subscribe 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.

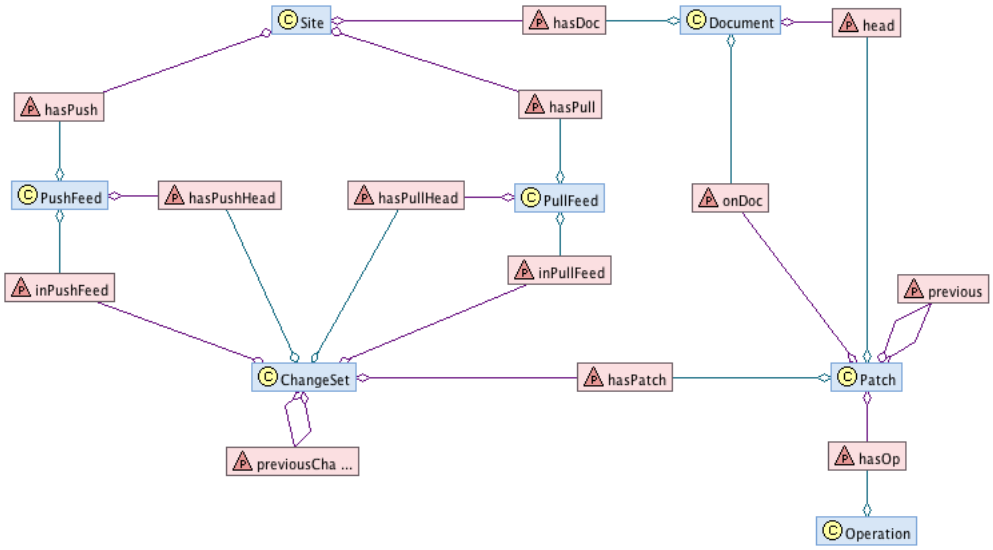

Figure 1. Shared Causal History Ontology.

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

The ontology defines basic concepts common to distributed version control systems 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.

- *Site*: a site has the following properties:
  - siteID: This attribute contains the identifier of the site.
  - hasPush, hasPull and hasDoc : The range of these properties are respectively a *PushFeed*, a *PullFeed* and a *Document*. A site has several PushFeeds, several PullFeeds and several Documents.
- *Document*: a document has the following 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 property:
  - 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: Points to the precedent executed *Patch* on the local site.
- *ChangeSet*: A set of patches. This concept 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 properties:
  - changSetID: A unique identifier of a *ChangeSet*.
  - hasPatch: Points to the *Patches* generated since the last push.
  - previousChangeSet: 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*.
  - 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 properties:
  - pullID: A unique identifier of the *PullFeed*.
  - hasPullHead: This property points to the last pulled *ChangeSet*.

The ontology is managed by the five operations mentioned earlier. The algorithms for these operations are presented in the following. These algorithms ensure that each site implements a causal reception (Rahhal et al., 2009). Consequently, the proposed framework ensures causality.

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(doc : String, docMod : String) :
  Patch(pid = concat(site.siteID, site.logicalClock++))
  foreach op ∈ diff(doc, docMod) do
    Operation(opid = concat(site.siteID, site.logicalClock++))
    hasOp(pid, opid)
  endfor
  previous(pid, doc.head)
  head(doc, pid)
  onDoc(pid, doc)
```

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

```
createPush(name : String, docs : set of Document ) :
  PushFeed(name)
  hasPush(site, name)
  call Push(name)
```

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

```
push(name : String) :
  ChangeSet(csid=concat(site.siteID, site.logicalClock++))
  inPushFeed(csid, name)
  published = {∃x∃y(Patch(x)∧Changeset(y)∧inPushFeed(y,name)∧hasPatch(y,x))}
  patches = { ∃x∃p (Patch(P)∧Document(x)∧onDoc(x,p))}
  foreach patch ∈ patches / published
    hasPatch(csid, patch)
  endfor
  previousChangeSet(csid, name.hasPushHead)
  hasPushHead(name, csid)
```

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

```
createPull(name : String, pushID : int) :
  PullFeed(name);
  call Pull(name);
```

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

```
pull(name : String) :
  cs = get(name.headPullFeed)
  while (cs ≠ null)
    CS' = {∃x(ChangeSet(x)∧inPushFeed(x,name))}
    if cs ∉ CS' then
      inPullFeed(cs, name)
      call Integrate(cs)
    endif
    cs = cs.previousChangeSet
  endwhile
```

The "Integrate" function in the *pull* algorithm is an Commutative Replicated Data Type (CRDT) algorithm (Weiss et al., 2010) (Weiss et al., 2009) (Preguica et al., 2009)

Divergence Metrics in SCHO

This section details the formal definition of existing divergence awareness metrics based on SCHO model. 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 formula:

- ***onSite(P,D,S):*** This means that a patch P belongs to a document D was generated on a site S.

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

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

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

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

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

State Treemap Divergence Awareness

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

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

$$LM(D, LS) \equiv \exists P \exists D \exists LS : Patch(P) \wedge Document(D) \wedge Site(LS) \wedge onSite(P, D, LS) \wedge \neg inPushFeed(P, LS)$$

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

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

- ***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.

$$PC(D, LS) \equiv \exists D \exists LS \exists RS : Document(D) \wedge Site(LS) \wedge Site(RS) \wedge (LS \neq RS) \wedge LM(D, LS) \wedge RM(D, LS)$$

- ***Need Update(LNU):*** There are PushFeed(s) in remote sites that were not pulled locally.

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

- ***Will Conflict (WC):***

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

- ***Locally Up To Date (UTD)***

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

Palantir Divergence Awareness

We define the divergence awareness of Palantir detailled in the table I using SCHO ontology.

- ***Populated (Pop):*** A document has been created on a site.

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

- ***Change in progress (CP):***

  This state is similar to the Locally-Modified or Remotely-Modified states already mentioned in State Treemap.

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

- ***Change Reverted:*** The document has returned to its original state.

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

- ***Severity Changed:***

  The number of patches has been done on a document.

$$\sum P : P \in \{LM(D, S)\}$$
$$\sum P : P \in \{RM(D, S)\}$$

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.

Concurrent modification Divergence Awareness

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 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.

$$CM \equiv \exists D \exists S \exists P1 \exists P2 : Document(D) \wedge Site(S) \wedge hasDoc(S, D) \wedge onDoc(P1, D) \wedge onDoc(P2, D) \wedge \neg previous(P1, P2)$$

Validation

In order to validate our approach we populated the SCHO ontology with the causal history data. We used *git* (Git, 2005) repositories. *git* is a distributed version control system that supports a multi-synchronous collaboration. *git* repositories have rich sets of data with different size that can be used to compute divergence metrics.

To use *git* data, First we had to inject the *git* history data into a triple store to populate our ontology. We used the Jena TDB (Jena, 2009) triple store, then we implemented a parser code which is responsible for the mapping between the concepts defined in *git* history and SCHO ontology. Figure 2 shows the history log of a sample *git* project.

We made the following assumptions when we parsed the *git* history:

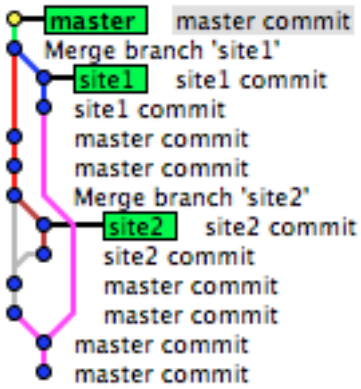

Figure 2. A sample project *git* history.

- 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 changeset we create a PullFeed and the corresponding Site.
- Each site represents one user.

So for the sample project we will have: three Sites, two PullFeeds, two PushFeeds and thirteen ChangeSets. Figure 3 shows the populated ontology resulting from parsing the *git* history. Figure 4 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 4.

Then we used the metrics described earlier to calculate the divergence between the sites using SPARQL (SPARQL, 2008) queries. For example the following query returns the state *Remotely Modified* for a ChangeSet *$CSid*.

```
SELECT ?pf  WHERE {
        MS2W:$CSid MS2W:inPullFeed ?pf .
        MS2W:$CSid MS2W:date ?date .
        ?pf MS2W:hasPullHead ?CSHead
        MS2W:?CSHead MS2W:date ?headDate .
        NOT EXISTS { MS2W:$CSid MS2W:published "true".}
        FILTER ( xsd:dateTime(?headDate) <= xsd:dateTime(?date) )
        }
```

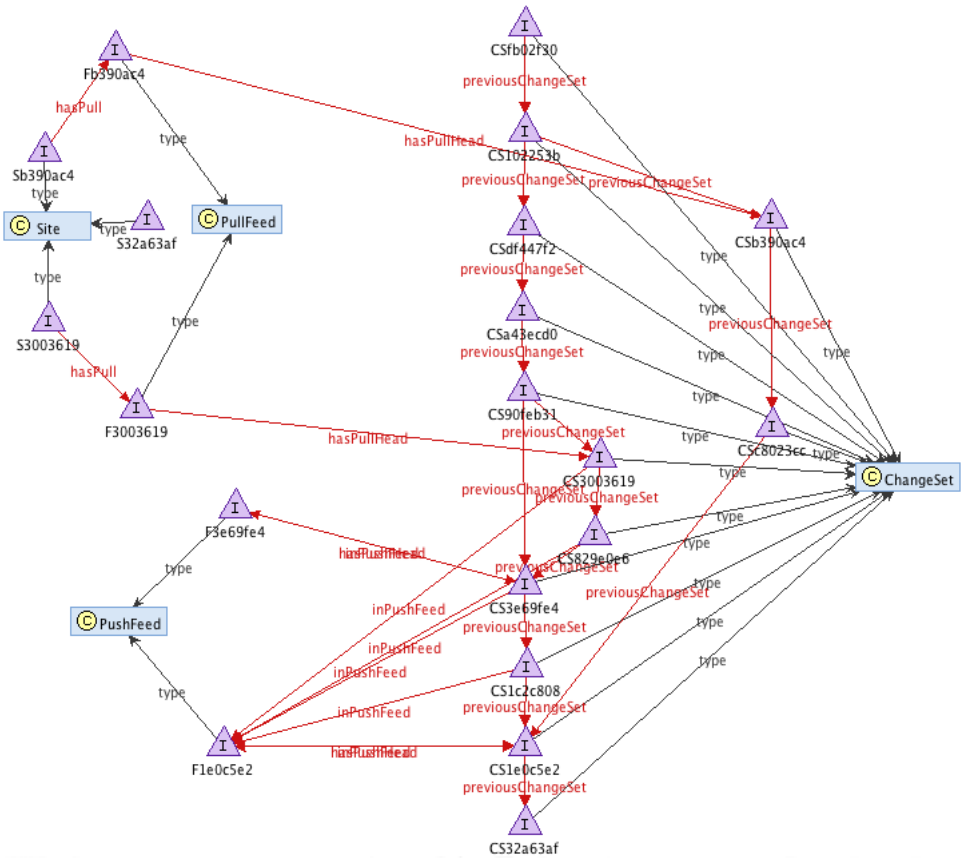

Figure 3. A sample project equivalent RDF graph.

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

```
SELECT ?pf ?date WHERE {
        MS2W:$CSid MS2W:inPushFeed ?pf .
        MS2W:$CSid MS2W:date ?date .
        FILTER ( xsd:dateTime(?date) <= \$date^^xsd:dateTime )
        }
```

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 II shows the resulted states for State Treemap and Palantir for our sample project.

We plotted the corresponding divergence graph. Figure 5 shows the results we found for the sample *git* project. We can observe clearly the divergence and convergence phases.

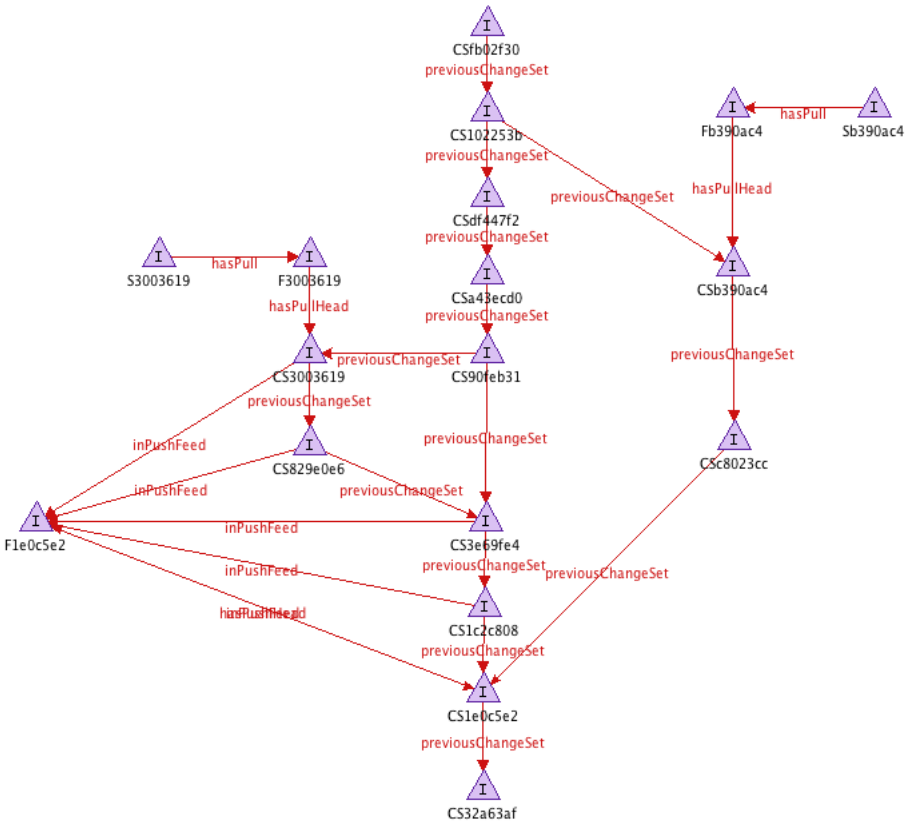

Figure 4. A sample project equivalent RDF graph without the concepts.

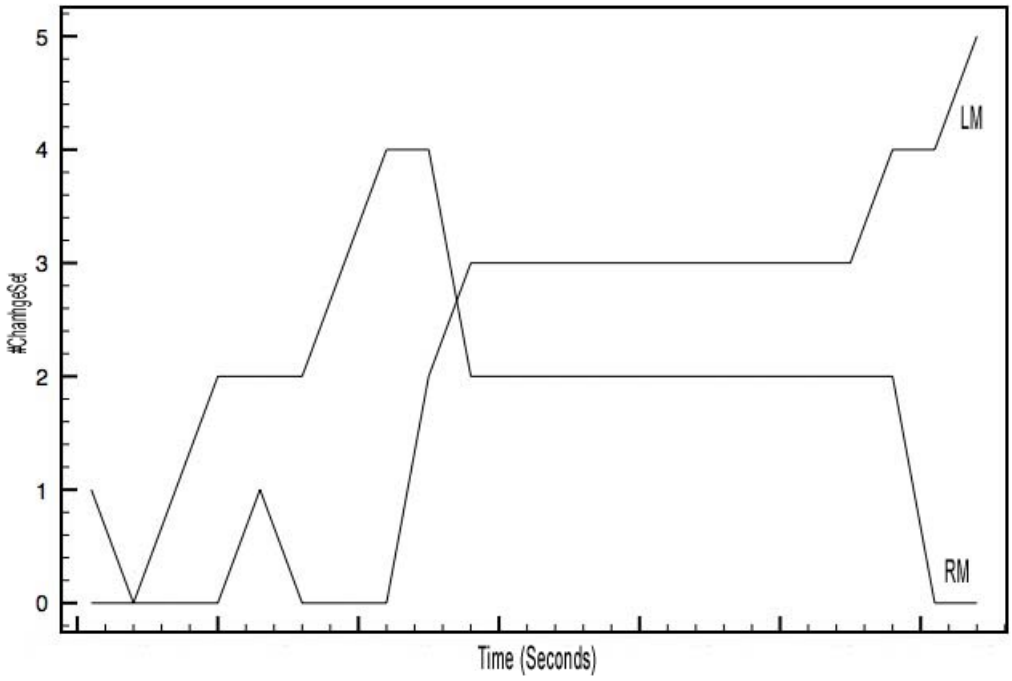

Figure 5. Divergence awareness for sample project.

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 II. Divergence awareness results for the sample project.

Then we used real *git* projects to validate the approach, such as reddit (Reddit, 2008), gollum (Gollum, 2010), CakePHP (CakePHP, 2005) and MongoDB (MongoDB, 2009). The reddit project has 424 changsets in its *git* history over 26 months. The gollum project has 554 changesets over 10 months. The CakePHP project has 7508 changesets over 64 months. The mongoDB has 11086 changesets over 38 months.

Table III shows the details of each project and the execution time for populating the ontology with the causal history of these projects that we used to validate our approach.

Project name	#ChangeSets	Project Lifespan (month)	Execution time (ms)
Sample project	13	-	2079
Reddit	424	26	3549
Gollum	554	10	7976
CakePHP	7508	64	384409
MongoDB	11086	38	783560

Table III. Execution time for populating the SCHO ontology.

Figures 6, 7 and 8 show the results obtained after calculating the divergence awareness metrics on the reddit, gollum and mongoDB projects respectively. The *Y-axis* represents the number of changesets, while the *X-axis* represents the time.

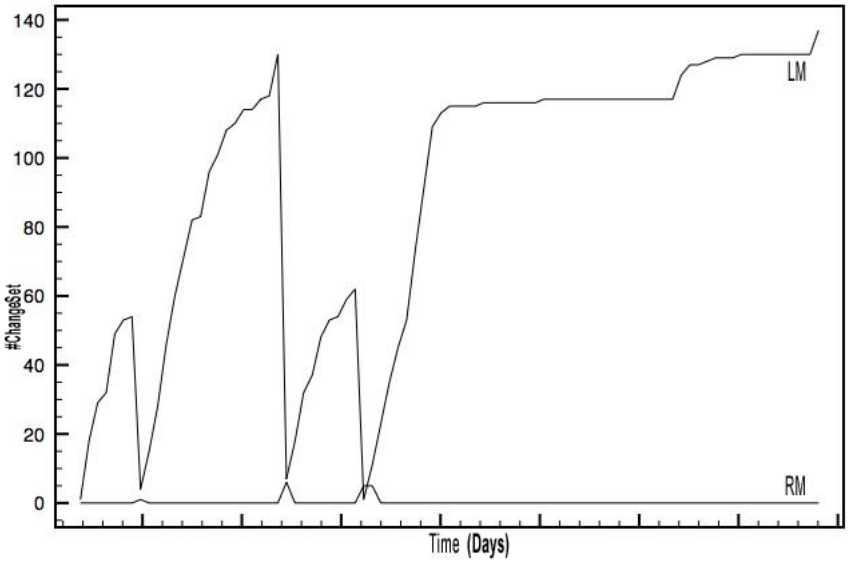

Figure 6. Divergence awareness for reddit project.

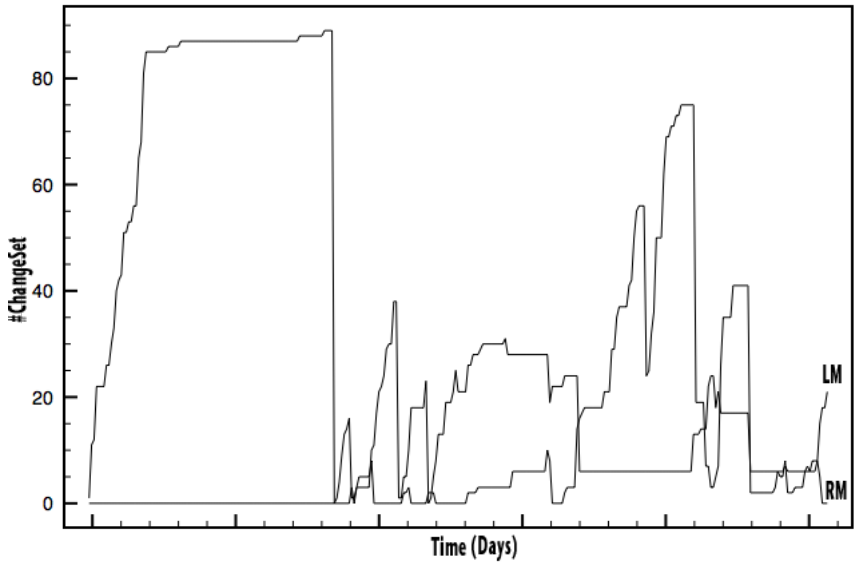

Figure 7. Divergence awareness for gollum project.

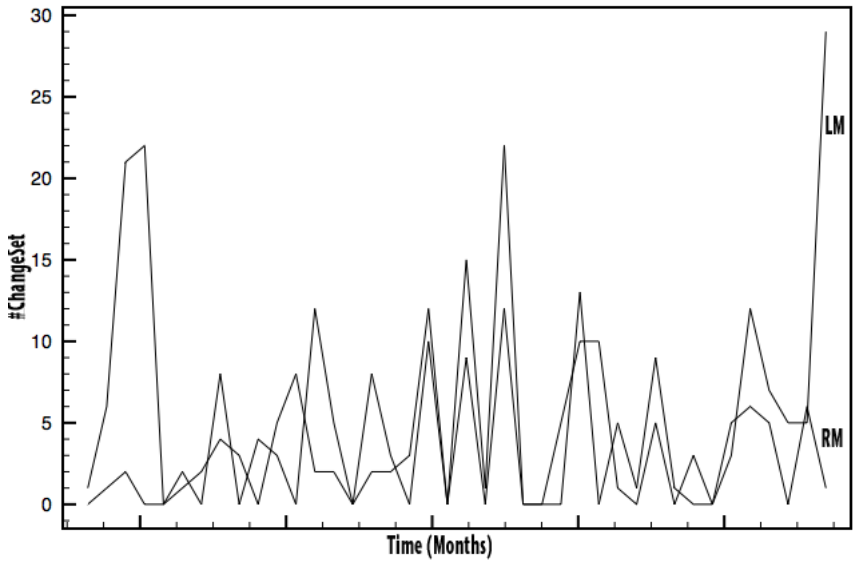

Figure 8. Divergence awareness for mongoDB project.

Conclusion

Building a system that computes divergence awareness can be a very challenging task. Different propositions have been made to compute divergence awareness, like StateTreemap, Palantir, etc. Such divergence metrics are highly dependent on applications and cannot be reused outside of their original scope. In this work, we proposed a general framework that allow to capture and to share causal histories. We expressed this model using semantic technologies. Next, we demonstrated how it is possible to compute all existing divergence metrics on this framework using semantic queries. Thus, divergence metrics are no more dependent of application models. We validated this result by transforming the distributed version control systems history into this ontology and computing the divergence metrics on them.

In the future we plan to embed SCHO ontology inside distributed version control systems in order to allow users and developers to take advantage of divergence awareness and semantic technologies. We also plan to analyze more systems to characterize divergence evolution in time.

References

Allen, L., G. Fernandez, K. Kane, D. Leblang, D. Minard, and J. Posner (1995): 'ClearCase MultiSite: Supporting Geographically-Distributed Software Development'. *Software Configuration Management: Scm-4 and Scm-5 Workshops: Selected Papers*.

CakePHP (2005): 'PHP framework'. http://www.cakephp.org/.

Dourish, P. (1995): 'The Parting of the Ways: Divergence, Data Management and Collaborative Work'. In: *4th European Conference on Computer Supported Cooperative Work*.

Ellis, C. A. and S. J. Gibbs (1989): 'Concurrency Control in Groupware Systems'. In: *SIGMOD Conference*, Vol. 18. pp. 399–407.

Git (2005): 'Fast Version Control System'. http://git-scm.com/.

Gollum (2010): 'A wiki built on top of Git'. https://github.com/github/gollum.git.

Jena (2009): 'Open source Semantic Web framework for Java'. http://openjena.org/.

Lamport, L. (1978): 'Times, Clocks, and the Ordering of Events in a Distributed System'. *Communications of the ACM*, vol. 21, no. 7, pp. 558–565.

Molli, P., H. Skaf-Molli, and C. Bouthier (2001): 'State Treemap: an Awareness Widget for Multi-Synchronous Groupware'. In: *Seventh International Workshop on Groupware - CRIWG*. IEEE Computer Society.

MongoDB (2009): 'Document oriented database'. http://www.mongodb.org/.

Papadopoulou, S., C. Ignat, G. Oster, and M. Norrie (2006): 'Increasing Awareness in Collaborative Authoring through Edit Profiling'. In: *IEEE Conference on Collaborative Computing: Networking, Applications and Worksharing - CollaborateCom 2006*. Atlanta, Georgia, USA.

Preguica, N., J. M. Marques, M. Shapiro, and M. Letia (2009): 'A Commutative Replicated Data Type for Cooperative Editing'. *2009 29th IEEE International Conference on Distributed Computing Systems*, pp. 395–403.

Rahhal, C., H. Skaf-Molli, P. Molli, and S. Weiss (2009): 'Multi-synchronous Collaborative Semantic Wikis'. In: *10th International Conference on Web Information Systems Engineering - WISE '09*, Vol. 5802 of *LNCS*. pp. 115–129, Springer.

Reddit (2008): 'Reddi - the voice of the internet'. http://www.reddit.com/.

Saito, Y. and M. Shapiro (2005): 'Optimistic replication'. *ACM Computing Surveys*, vol. 37, no. 1, pp. 42–81.

Sarma, A., Z. Noroozi, and A. V. D. Hoek (2003): 'Palantr: Raising Awareness among Configuration Management Workspaces'. pp. 444–454.

SPARQL (2008): 'SPARQL Protocol and RDF Query Language'. http://www.w3.org/TR/rdf-sparql-query/.

Sun, C., X. Jia, Y. Zhang, Y. Yang, and D. Chen (1998): 'Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems'. *ACM Transactions on Computer-Human Interaction (TOCHI)*, vol. 5, no. 1, pp. 63–108.

Weiss, S., P. Urso, and P. Molli (2007): 'Wooki: a P2P Wiki-based Collaborative Writing Tool'. In: *Web Information Systems Engineering*. Nancy, France.

Weiss, S., P. Urso, and P. Molli (2009): 'Logoot : a Scalable Optimistic Replication Algorithm for Collaborative Editing on P2P Networks'. In: *International Conference on Distributed Computing Systems (ICDCS)*. IEEE.

Weiss, S., P. Urso, and P. Molli (2010): 'Logoot-Undo: Distributed Collaborative Editing System on P2P Networks'. *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 8.

A SCHO Ontology described in OWL

```
Namespace(rdf    = <http://www.w3.org/1999/02/22-rdf-syntax-ns#>)
Namespace(owl    = <http://www.w3.org/2002/07/owl#>)
Namespace(xsd    = <http://www.w3.org/2001/XMLSchema#>)
Namespace(rdfs   = <http://www.w3.org/2000/01/rdf-schema#>)
Namespace(a      = <http://www.semanticweb.org/ontologies/2009/4/scho.

Ontology( <http://www.semanticweb.org/ontologies/2009/4/scho.owl>

 Class(a:ChangeSet partial
  owl:Thing)
 Class(a:Document partial
  owl:Thing)
 Class(a:Operation partial
  owl:Thing)
 Class(a:Patch partial
  owl:Thing)
 Class(a:PullFeed partial
  owl:Thing)
 Class(a:PushFeed partial
  owl:Thing)
 Class(a:Site partial
  owl:Thing)
 Class(owl:Thing partial)

 ObjectProperty(a:hasDoc
  domain(a:Site)
  range(a:Document))
 ObjectProperty(a:hasOp
  domain(a:Patch)
  range(a:Operation))
```

```
  ObjectProperty(a:hasPatch
   domain(a:ChangeSet)
   range(a:Patch))
  ObjectProperty(a:hasPull
   domain(a:Site)
   range(a:PullFeed))
  ObjectProperty(a:hasPullHead
   domain(a:PullFeed)
   range(a:ChangeSet))
  ObjectProperty(a:hasPush
   domain(a:Site)
   range(a:PushFeed))
  ObjectProperty(a:hasPushHead
   domain(a:PushFeed)
   range(a:ChangeSet))
  ObjectProperty(a:head
   domain(a:Document)
   range(a:Patch))
  ObjectProperty(a:inPullFeed
   domain(a:ChangeSet)
   range(a:PullFeed))
  ObjectProperty(a:inPushFeed
   domain(a:ChangeSet)
   range(a:PushFeed))
  ObjectProperty(a:onDoc
   domain(a:Patch)
   range(a:Document))
  ObjectProperty(a:previous
   domain(a:Patch)
   range(a:Patch))
  ObjectProperty(a:previousChangeSet
   domain(a:ChangeSet)
   range(a:ChangeSet))

)
```