**Tenth Int. Workshop on Generalized Nets**
**Sofia, 5 December 2009, 14-21**

# Generalized Net Models of Conflict Resolution Approaches in Version Control Systems. Part 2: Copy-Modify-Merge

**Vassia Atanassova[1,2] and Peter Georgiev[2]**
[1] – Institute of Information Technologies, Bulgarian Academy of Sciences
"Acad. G. Bonchev" Street, block 2, Sofia-1113, Bulgaria
[2] – Centre of Biomedical Engineering, Bulgarian Academy of Sciences
"Acad. G. Bonchev" Street, block 105, Sofia-1113, Bulgaria
e-mails: *vassia.atanassova@gmail.com , prg@mail.bg*

**Abstract:** In the present research we develop Generalized Net (GN) models of the two core mechanisms for edit conflict resolution, implemented in various Version Control Systems (VCS). The hitherto constructed GN models allow visual comparison between both mechanisms, but only once we have a fully functional GN simulator, we would be able to run a simulation and draw the functional comparison that may assist the VCS developers for making the right choices when it comes to dealing with concurrent access.

**Keywords:** Generalized nets, Version control system, Concurrent access, Conflict resolution

## 1 Introduction

The first part of this research [2] dealed with the Lock-Modify-Unlock (LMU) approach to edit conflict resolution in VCS. It laid the basis of the second part by providing the basic definitions and arguments for choosing between LMU and the other approach, namely Copy-Modify-Merge (CMM) that is to be discussed here in more details.

To summarise, in the CMM approach (illustrated on Figure 1) each user accesses one and the same file from the repository and creates a local personal working copy. Thus, users are able to work simultaneously and independently, modifying their private copies. Finally, the system attempts to resolve eventual edit conflicts by merging the versions. into a new, final version. If it is not capable of doing so automatically, users are responsible for making it happen correctly.

For instance, in MediaWiki (the software that powers Wikipedia) if the page is divided into sections and different users simultaneously edit different sections, the system (with varying performance) is capable of merging their edits without causing a conflict. However, if overlapping of editing areas occurs, for instance edits the whole page, or both users edit the same section, or both users edit the whole page, then an edit conflict usually occurs and manual conflict resolution is necessary. In a variation of this approach, realized in other wiki platforms (like DocuWiki), a warning message is displayed if a user opens for editing a page that has already been opened by another user. Yet, this warning has only precautionary, rather than prohibitive, purpose and it may again lead to edit conflict.

The CMM mechanism may sound a bit chaotic, but in practice, it often runs smoothly. Users can work in parallel, never waiting for one another and the amount of time it takes to resolve conflicts is usually far less than the time lost by the locking system in the LMU approach.

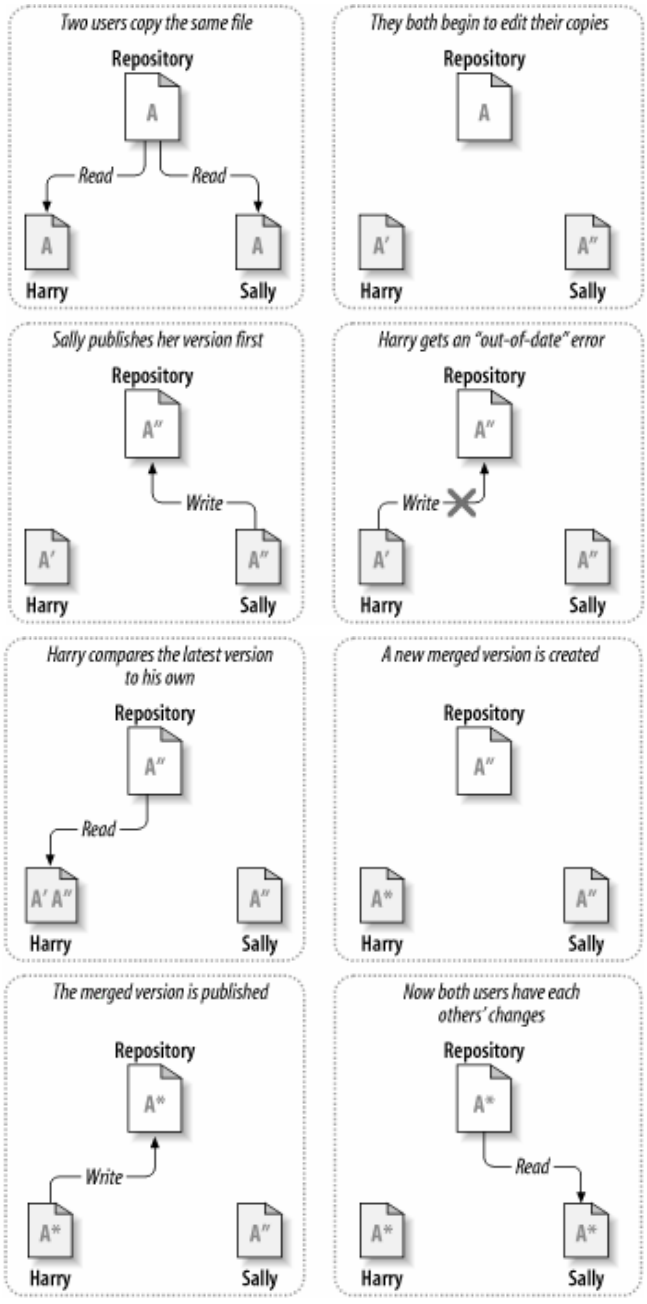


*Figure 1. Illustration of the CMM approach, [3]*

## 2 Generalized Net Model of the CMM Approach

The first part of the research contained all the notations used within the generalized net [1] model of the CMM approach to conflict resolution. However, certain improvements and streamlinings of the model structure, done after submitting the first part of the research, led to the reduction of some of the provisioned places and/or changing their names to more

intuitive ones. So repeating and updating this part here is not only helpful for the reader, but also necessary.

The GN model contains static components, called places and transitions. All transitions are noted by T, labeled with letter L or C (for LMU or CMM models, respectively) and the transition's serial number from left to right. All places, except of DB and ID, are labeled with the respective actions that occur in them, and again indexed by L or C. Both models inevitably contain common places (Open, Edit, Cancel, Save, Close), but the second one contains two additional places (Conflict and Merge). Places DB (standing for the database or repository) and ID (generator of version identifiers) are left without an index, since they do not carry semantic difference between the models.

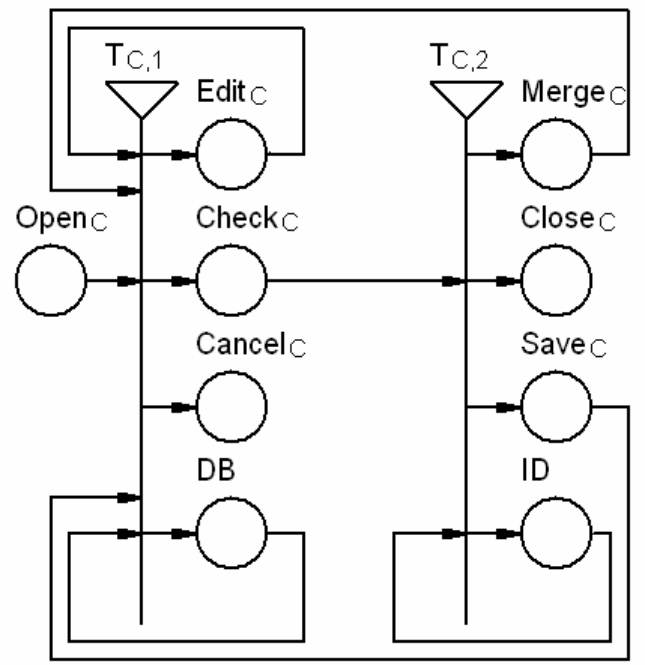


*Figure 2. GN model of the CMM approach*

Like the GN model in the first part of the research, the one here also operates with three types of tokens, which can split and merge, thus carrying the dynamic nature of the models:

- $\varepsilon$-token, standing for the editors (users), who access (open) a file. This token initially enters the net through place Open.
- $\delta$-token, staying only in place DB, and representing the database of files. The separate instances of files are tokens, noted by $\varphi$ and once they split from the $\delta$-token, they may move around the net.
- $\iota$-token, staying only in place ID, and representing the generator of version identifiers. The separate identifiers of the file versions are natural numbers, which in the GN model are presented as tokens, noted by $v$. Once they split from the $\iota$-token, they may move around the net.

The logic of the GN model is contained in the index matrices of the transitions, which are noted by M and labeled with the indices of the respective transitions. All predicates in a given index matrix, which differ from true and false, are noted by P and labeled with the model's type (in this case, C), the transition's serial number and the predicate's serial number. The GN model is presented on Figure 2.

Every $\varepsilon$-token enters the net via place $\mathsf{Open}_L$ with an initial characteristic: *"User ID (Username or IP-address) of the editor. Requested file. Timestamp of the moment of access to the file"*.

Token $\delta$ permanently stays in place DB with characteristic: *"List of files in the database, containing: (1) File identifier. (2) Identifier and timestamp of the current file version. (3) List of file copies, associated with a copy ID, user ID, and ID and timestamp of the file version that was current at the moment of retrieving the copy"*.

| File (1) | Current version ID and timestamp (2) | List of copy version IDs, timestamps and users (3) |
|---|---|---|
| $file_1$ | $id_1$, $ts_1$ | [$copy_1$, $user_{1,1}$, $id_{1,1}$, $ts_{1,1}$],<br>[$copy_2$, $user_{1,2}$, $id_{1,2}$, $ts_{1,2}$],<br>... |
| … | … | … |
| $file_k$ | $id_k$, $ts_k$ | [$copy_1$, $user_{k,1}$, $id_{k,1}$, $ts_{k,1}$],<br>[$copy_2$, $user_{k,2}$, $id_{k,2}$, $ts_{k,2}$],<br>... |
| … | … | … |
| $file_n$ | $id_n$, $ts_n$ | [$copy_1$, $user_{n,1}$, $id_{n,1}$, $ts_{n,1}$,],<br>[$copy_2$, $user_{n,2}$, $id_{n,2}$, $ts_{n,2}$],<br>... |

For example, if several users have started editing copies of one and the same version of a file, and in the meanwhile one of these users saves their edit first, this will change the current version ID and timestamp in (2), but this will not affect the version IDs and timestamps for the rest concurrent editors in (3). Once any of them attempts to save their edit, the difference between the current (already changed) file version ID and the version ID of their copy will trigger the system's conflict resolution mechanism and will either merge both versions automatically (if possible), or it will prompt the user for the need to manually resolve the edit conflict.

While token $\delta$ represents the whole database (repository), the separate instances of files from the database will be represented as $\varphi$-tokens.

Token $\iota$ permanently stays in place ID with characteristic: *"Number of file revisions made in the system since a given moment"*. In other words, this token plays the role of a generator of subsequent natural numbers which identify the revisions that occur in all files within the database. By default, this number is 0, but if a particular simulation should start from a particular moment from the system's functioning, the respective number of file revisions made by this moment should be associated with the token.

While token $\iota$ represents the generator of identifiers, the separate instances of revision identifiers will be represented as $v$-tokens.

The GN model consists of two transitions and nine places. Let us describe them in details.

$$T_{C,1} = <\{\mathrm{Edit}_C, \mathrm{Merge}_C, \mathrm{Open}_C, \mathrm{Save}_C, \mathrm{DB}\},$$
$$\{\mathrm{Edit}_C, \mathrm{Check}_C, \mathrm{Cancel}_C, \mathrm{DB}\}, M_{C,1} >$$

$M_{C,1} =$

| | $Edit_C$ | $Check_C$ | $Cancel_C$ | DB |
|---|---|---|---|---|
| $Edit_C$ | $P_{C,1,2}$ | $P_{C,1,4}$ | $P_{C,1,3}$ | $P_{C,1,3}$ |
| $Merge_C$ | *true* | *false* | *false* | *false* |
| $Open_C$ | $P_{C,1,1}$ | *false* | *false* | *false* |
| $Save_C$ | *false* | *false* | *false* | *true* |
| DB | $P_{C,1,1}$ | $P_{C,1,5}$ | *false* | *true* |

where

- $P_{C,1,1}$ = *"The user is provided with a copy of the file from the database, opened in edit mode."*
- $P_{C,1,2}$ = *"The user continues editing the file."*
- $P_{C,1,3}$ = *"The user stops editing the file and does not want to save the edit."*
- $P_{C,1,4}$ = *"The user stops editing the file and wants to save the edit."*
- $P_{C,1,5}$ = *"There is edit conflict detected."*

When predicate $P_{C,1,1}$ is true, token $\varepsilon$ leaves place $Open_C$ and enters place $Edit_C$. Simultaneously token $\delta$ in place DB splits to the same token $\delta$ and a new token $\varphi$, which represents a copy of the file from the database, requested by the current user for editing. Token $\varphi$ enters place $Edit_C$ with current characteristic: *" {1} File ID. {2} Version ID, timestamp and contents of the current file version"*.

Token $\delta$ makes one loop and returns to place DB, where it updates its characteristic table by adding a new entry in the third column for the newly made copy of the requested file $\varphi$, as characterised by its current file version ID, the moment (timestamp) of the copy and the user who requested the copy, as symbolised by $\varepsilon$.

So, when the $\varepsilon$-token from place $Open_C$ and the $\varphi$-token from place DB enter place $Edit_C$, they merge together under the name of the $\varepsilon$-token, which thus obtains the characteristic: *" {1} User ID. File ID. {2} Version ID, timestamp and contents of the most recent version of the file. {3} Timestamp and contents of the file copy."* Of course, in the very first moment, before the user starts making changes, the contents of the file copy will be identical to the contents of the most recent version of the file. Their timstamps, however, will be different, and the file copy will not be associated with a version ID.

Predicate $P_{C,1,2}$ being true means that the user keeps editing the retrieved personal copy of the file, i.e. the $\varepsilon$-token keeps looping in place $Edit_C$. Taking account of the timestamp when the file copy was retrieved is important, since it allows comparing the local copy with the one, currently stored in the database. Based on this timestamp, the system may feature intermediate notification of the users for changes in the file, they have been working on. On each loop in place $Edit_C$, the $\varepsilon$-token updates its characteristics, especially in its last part "Contents of the file copy".

Predicate $P_{C,1,2}$ being false means that either predicate $P_{C,1,3}$ or $P_{C,1,4}$ may become true: the user decides to cease editing the file. Which of both becomes true determines whether the editor wants to save his/her changes or not. If the user wants to cancel editing, without saving the changes, predicate $P_{C,1,3}$ becomes true and the $\varepsilon$-token from place $Edit_C$ splits back into two token: the same $\varepsilon$-token and the initial $\varphi$-token. The $\varepsilon$-token enters place $Cancel_C$ without any characteristic, while the $\varphi$-token enters place DB and merges there with the $\delta$-token, updating the characteristic table by removing in (3) the corresponding entry for the copy of the respective file, requested by the respective user. Once removed, the entry will not affect the experience of the rest editors by triggering eventual edit conflicts.

Eventually, the user may decide to save his/her changes, which corresponds to predicate $P_{C,1,4}$ becoming true, which triggers the transfer of the $\varepsilon$-token from place $Edit_C$ to

place $\mathsf{Conflict}_C$ with its latest characteristics. But here, it becomes of major importance what the value of predicate $P_{C,1,5}$ will be. Predicate $P_{C,1,5}$ corresponds to the system check of whether an edit conflict has occurred for the discussed file copy, or not. This check is easily done within place DB, owing to the fact that the $\delta$-token's characteristic table maintains information not only of the file version identifiers, but about the timestamps of the current file version and the timestamps of each of the produced copies of the file, as opened for editing. When a new version of a file is successfully saved in the database, the current file version's timestamp is updated, and if copies of the file have been already produced, their timestamps are smaller than the updated current version's timestamp, hence the predicate $P_{C,1,5}$ becomes true. This leads to the $\delta$-token in place DB splitting to two tokens: the original $\delta$-token, carrying the full information about the system's database, and a new token $\varphi_{cur}$ that transfers to place $\mathsf{Conflict}_C$ with the characteristic "Current version ID, timestamp and contents of the file". There, the $\varphi_{cur}$-token unites with the $\varepsilon$-token, which arrived due to true value of predicate $P_{C,1,4}$. Thus, place $\mathsf{Conflict}_C$ corresponds to the moment when the system detects an edit conflict and requires the user to personally and manually resolve it.

Some version control systems are implemented in a way that allows automatic edit conflict resolution. The particularities of this procedure is not within the scope of this research, and for our modelling needs only a Yes/No output is enough. However, elaborating on the precise mechanism of automatic resolution is an interesting future step of extending the GN model. A possible approach is comparing the areas of the document where the concurrent editors have made changes. If these areas do not intersect, for instance one of the users has edited the head of the document, and the other has edited the tail, then the system is able to automatically resolve the conflict, without even notifying the second user of the conflict. This is a purely technical approach to conflict resolution, and in general it is not possible to take account of the *semantics* of each of the concurrent edits.

So, in place $\mathsf{Conflict}_C$ the newly merged $\varepsilon$-token obtains the combined characteristic: *"{1} User ID. File ID. {2} Version ID, timestamp and contents of the file version that <u>used to be current</u> in the moment of producing the copy. {3} Timestamp and contents of the file copy. {4} Version ID, timestamp and contents of the <u>current</u> file version."* In practice, this corresponds to the moment, when the user is notified of the edit conflict and provided with all or a part of the data, contained in the current characteristics of the $\varepsilon$-token, depending on the user interface of the particular version control system. For instance, the edit conflict notification screen in MediaWiki offers the user with three pieces of information: the contents of the current file version {4}, the contents of his/her copy of file, as edited hitherto {3} and a difference between revisions, called *difflink*. As we will see below, the user may either approve the interim version, or may wish to continue editing, merging his/her own edit with the latest one.

Place $\mathsf{Conflict}_C$ is the only place that links the both transitions in the model, i.e. turns to be an output place for $T_{C,1}$ and an input place for transition $T_{C,2}$ that is represented as:

$$T_{C,2} = < \{\mathrm{Conflict}_C, \mathrm{ID}\}, \{\mathrm{Merge}_C, \mathrm{Close}_C, \mathrm{Save}_C, \mathrm{ID}\}, M_{C,2} >$$

$$M_{C,2} = \begin{array}{c|cccc} & \mathrm{Merge}_C & \mathrm{Close}_C & \mathrm{Save}_C & \mathrm{ID} \\ \hline \mathrm{Conflict}_C & P_{C,2,1} & P_{C,2,2} & P_{C,2,2} & \mathit{false} \\ \mathrm{ID} & \mathit{false} & \mathit{false} & P_{C,2,2} & \mathit{true} \end{array}$$

where

- $P_{C,2,1}$ = *"The user continues editing."*
- $P_{C,2,2} = \neg P_{C,2,1}$

If predicate $P_{C,2,1}$ is true, this corresponds to the scenario when the user, being notified of the edit conflict, has chosen to continue editing. The particular reasons for this decision might be various, including or not qualitative evaluation of the interim changes. For instance, the user not only wants to edit the originally accessed version of the file, but s/he also disagrees with the interim changes and wants to partially or fully revert them; or the user is in accord with the interim changes but his/her own changes are more extensive, sophisticated or valuable in a way. Whatever the particular motivation, the result of the predicate $P_{C,2,1}$ being true is that the $\varepsilon$-token from place $\mathsf{Conflict}_C$ transfers to place $\mathsf{Merge}_C$, where, as we know from the first predicate matrix $M_{C,1}$, it unconditionally transfers to place $\mathsf{Edit}_C$.

In the opposite case, if predicate $P_{C,2,1}$ is false, i.e. predicate $P_{C,2,2}$ is true, this corresponds to the situation when the system allows the user to save his/her edit, since no edit conflict has been detected. How exactly will this happen in terms of the generalized net, depends on the value of the predicate $P_{C,1,5}$ from above.

- If predicate $P_{C,1,5}$ was false, i.e. no conflict has been detected, the user is technically unprevented to save his/her edit. Hence, the $\varepsilon$-token from place $\mathsf{Conflict}_C$ splits to two tokens: token $\varepsilon$ that represents the editor and token $\varphi_{new}$ that represents the newly produced version of the file. At the same time, the $\iota$-token in place ID also splits in two: the same $\iota$-token and a new $\nu$-token that represents the consequent version identifier. The $\iota$-token keeps looping in place ID with is characteristic incremented with 1. The $\nu$-token moves from place ID to place $\mathsf{Save}_C$ where it unites with token $\varphi_{new}$, that obtains the characteristic: *"User ID. File ID. Version ID. Timestamp. Editted file contents"* where the version ID is the one obtained from the characteristic of the $\nu$-token and the timestamp corresponds to the current moment of saving the edit. Further, as we know from the first index matrix $M_{C,1}$, the transfer from place $\mathsf{Save}_C$ to place DB is unconditionally true, which appends this data as a new entry in the file history in the database. The token $\varepsilon$ itself transfers from place $\mathsf{Conflict}_C$ to place $\mathsf{Close}_C$ with a slightly shorter characteristic: *"User ID. File ID. Version ID. Timestamp"* which in practice corresponds to closing the edit session and adding a new entry to the list of user contributions (in case that the system keeps track of these).
- If predicate $P_{C,1,5}$ was true, i.e. a conflict has been detected, yet the user does not want to continue editing, this reflects the situation when the user either cancels editing at this (later) stage, or s/he approves the interim edit and ignores his/her own changes of the local copy. In terms of generalized nets, it means that the $\varepsilon$-token from place $\mathsf{Conflict}_C$ splits to two tokens: token $\varepsilon$, which transfers to place $\mathsf{Close}_C$ without any characteristic, and token $\varphi_{new}$ which transfers to place $\mathsf{Save}_C$ without the characteristic: *"Removal from the database of the corresponding entry for the current copy of the file"*. Once removed from the characteristic table, this file copy will no more affect the rest of the opened copies and trigger eventual edit conflicts. As we already know from the first predicate index matrix, the transfer from place $\mathsf{Save}_C$ to place DB is unconditionally true.

# 4 Discussions and conclusion

The presented GN model describes the Copy-Modify-Merge approach to the file sharing problem in version control systems. Surprisingly, the model is not more complicated than the GN model of the Lock-Modify-Unlock approach, and despite the differences in the tokens' movement around the net, both structures are almost the same. The third article in this series

will be devoted to comparing the results of testing both models with real data on the GN software simulator that is in an advanced stage of development.

## Acknowledgments

Figure 1 is taken from the book "Version Control with Subversion" by B. Collins-Sussman, B. W. Fitzpatrick, C. M. Pilato, published under Creative Commons Attribution License, http://creativecommons.org/licenses/by/2.0/.

## References

[1] Atanassov K. (2007), On Generalized Nets Theory, Sofia, "Prof. M. Drinov" Publ. House.

[2] Atanassova V. (2009), Generalized Net Models of Conflict Resolution Approaches in Version Control Systems. Part 1: Lock-Modify-Unlock, presented at International Workshop on Intuitionistic Fuzzy Sets and Generalized Nets, 16 Oct. 2009, Warsaw, Poland (in press)

[3] Collins-Sussman B., Fitzpatrick B. W., Pilato C. M. (2004), Version Control with Subversion, O'Reilly Media, http://svnbook.red-bean.com/.