Developments in Fuzzy Sets, Intuitionistic Fuzzy Sets, Generalized Nets and Related Topics. Volume I: Foundations

Developments in Fuzzy Sets, Intuitionistic Fuzzy Sets, Generalized Nets and Related Topics Volume II: Applications

Editors

Krassimir T. Atanassov
Michał Baczyński
Józef Drewniak
Janusz Kacprzyk
Maciej Krawczak
Eulalia Szmidt
Maciej Wygralak
Sławomir Zadrożny

Editors

Krassimir T. Atanassov
Władysław Homenda
Olgierd Hryniewicz
Janusz Kacprzyk
Maciej Krawczak
Zbigniew Nahorski
Eulalia Szmidt
Sławomir Zadrożny

SRI PAS IBS PAN

Developments in Fuzzy Sets, Intuitionistic Fuzzy Sets, Generalized Nets and Related Topics Volume II: Applications

Systems Research Institute
Polish Academy of Sciences

Developments in Fuzzy Sets, Intuitionistic Fuzzy Sets, Generalized Nets and Related Topics Volume II: Applications

Editors

Krassimir T. Atanassov
Władysław Homenda
Olgierd Hryniewicz
Janusz Kacprzyk
Maciej Krawczak
Zbigniew Nahorski
Eulalia Szmidt
Sławomir Zadrożny

IBS PAN SRI PAS

Systems Research Institute
Polish Academy of Sciences
Newelska 6, 01-447 Warsaw, Poland
www.ibspan.waw.pl

ISBN 9788389475305

Generalized net models of conflict resolution approaches in version control systems. Part 1: lock-modify-unlock

Vassia Atanassova
Institute of Information Technologies – Bulgarian Academy of Sciences
"Acad. G. Bonchev" Street, block 2, Sofia-1113, Bulgaria
Centre of Biomedical Engineering – Bulgarian Academy of Sciences
"Acad. G. Bonchev" Street, block 105, Sofia-1113, Bulgaria
vassia.atanassova@gmail.com

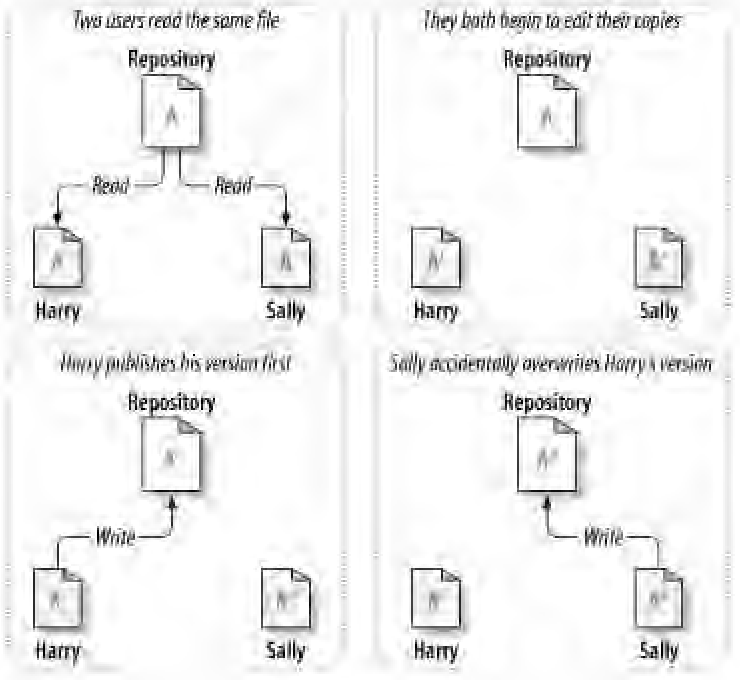
Abstract

In the present article 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

Version control (also called revision control, source control or source code management) is the management of changes to documents, programs, and other information stored as computer files. It is typically needed and used in multi-user environments like software development platforms, wiki-s and content management systems, where a team of people may collaborate over the same set of files. Each revision (change) on a file from the set is uniquely identified, accessible over time and comparable with the rest file revisions. It is also associated with a timestamp, username (or IP-address) of the person making the change, as well as other optional data. Revisions can be compared, restored, and with some types of files, merged.



Two users read the same file
Repository
Read
Read
Harry
Sally
They both begin to edit their copies
Repository
Harry
Sally
Harry publishes his version first
Repository
Write
Harry
Sally
Sally accidentally overwrites Harry's version
Repository
Write
Harry
Sally

Figure 1: Edit conflict situation

Although the concept of versioning exists in different software products, version control systems (VCS) are most commonly stand-alone applications and during the process of their development and/or functioning, an important matter to deal with is the concurrent access by multiple users, that may easily result in edit conflicts and accidentally overwriting and losing information. Edit conflict occurs when one user attempts to edit a document, but upon trying to save the new version, another person has already modified the document in the intervening time period, thus causing a difference between the attempted edit and the already-made edit that must be resolved manually (see Figure 1 and Acknowledgements section).

2 Conflict resolution mechanisms

The problem with the concurrent write access is fundamental for all version control systems and systems with shared file repositories. With certain variations, there are two general approaches to handle this problem:

- Lock – Modify – Unlock (LMU), and
- Copy – Modify – Merge (CMM).

The LMU mechanism is line-oriented and is considered pessimistic, while the CMM one is graph-oriented and is considered optimistic. Each of them has their benefits and drawbacks and the appropriateness of their choice depends on several factors like the type of files in the repository (binary files cannot be contextually merged, compared to line-based text files), number of files in the repository, size of the contributing community, average time of producing a revision, average volume of changes made, and recorded frequency of edit conflicts occurred.

2.1 Lock-modify-unlock

LMU (Figure 2) is a simple and popular approach to preventing problems of concurrent access is file locking so that only one developer at a time has write access to the central repository copies.

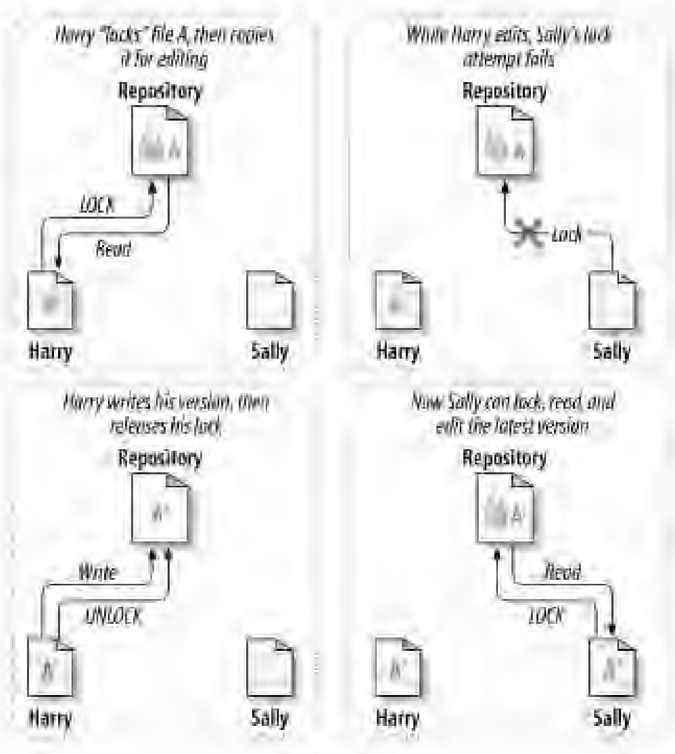

Harry "locks" file A, then copies it for editing
Repository
A
LOCK
Read
Harry
Sally
While Harry edits, Sally's lock attempt fails
Repository
A
Lock
Harry
Sally
Harry writes his version, then releases his lock
Repository
A'
Write
UNLOCK
A'
Harry
Sally
Now Sally can lock, read, and edit the latest version
Repository
A'
Read
LOCK
A'
A'
Harry
Sally

Figure 2: Lock - Modify - Unlock

Opening a file for editing is called check out and once performed, it prevents other users from changing the file, but only allows read access, until the editor checks in the updated version and unlocks the file, or cancels the check-out.

Along with the benefit of its easy implementation, this approach has its drawbacks too. It can be restrictive and result in waste of time and productivity. Sometimes the first contributor forgets about having locked a file and thus may unnecessarily prevent other users from editing. This may require administrative intervention for unlocking the file. Another hindrance of this mechanism is that it may cause unnecessary versioning of a file, in the cases when the users' edits do not overlap. And yet, file locking may create a "false sense of security" in cases when different users lock and edit different documents, which are dependent on one another, and changes made to each are semantically incompatible.

2.2 Copy-modify-merge

In the alternative approach of CMM (Figure 3) each user accesses 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.

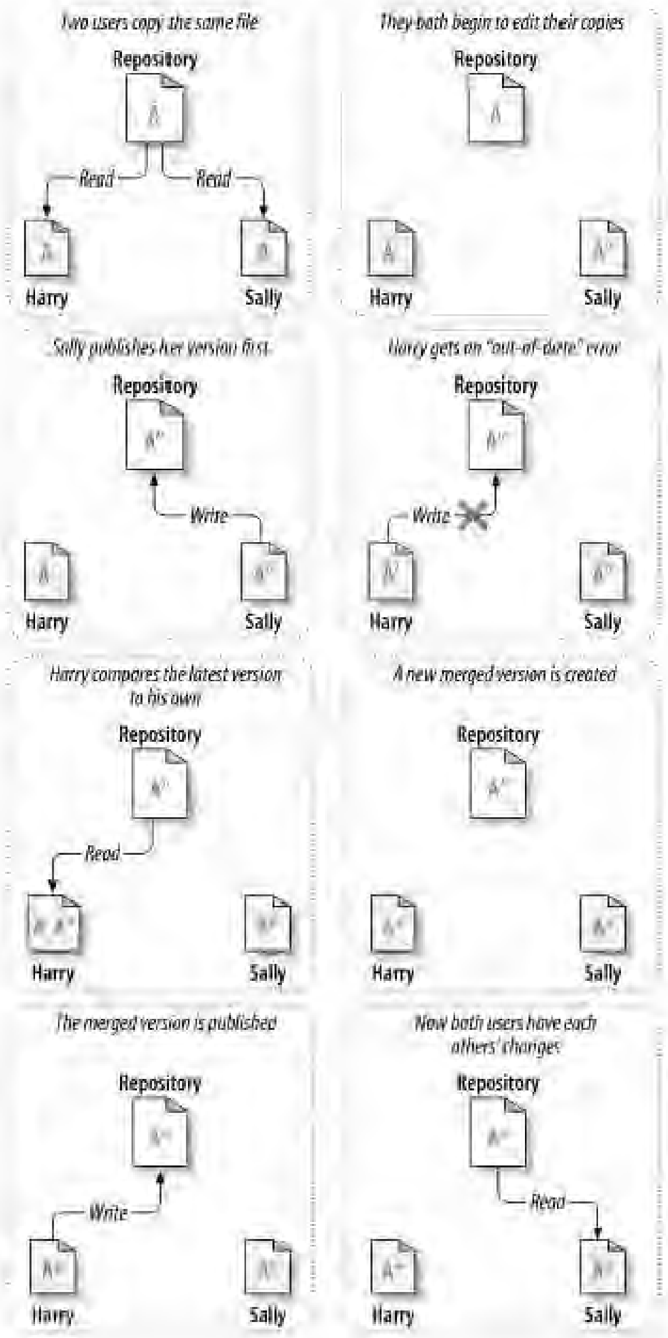

Two users copy the same file
Repository
Read
Read
Harry
Sally
They both begin to edit their copies
Repository
Harry
Sally
Sally publishes her version first
Repository
Write
Harry
Sally
Harry gets an "out-of-date" error
Repository
Write
Harry
Sally
Harry compares the latest version to his own
Repository
Read
Harry
Sally
A new merged version is created
Repository
Harry
Sally
The merged version is published
Repository
Write
Harry
Sally
Now both users have each others' changes
Repository
Read
Harry
Sally

Figure 3: Copy - Modify - Merge

3 Generalized net model of LMU approach

The focus of the present article is the design of a generalized net model, for the LMU versioning model. In a second part of this research, the CMM model will be described. Generalized nets (GNs) are a powerful tool for modelling parallel and concurrent processes with well developed mathematical apparatus behind them. Once we have a fully functional GN simulator, we would be able to run both models in a parallel simulation and draw the functional comparison between them. The analysis of the results may assist VCS developers for making the right choices when it comes to choosing an approach to implement. By further elaborating these relatively simple GN models by using hierarchical operators, different variants and extra features of both approaches may be tested in advance, and implemented if deemed appropriate.

First of all, a small remark on the notation conventions will be given and it will apply for the second part of this research.

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, Wait, Save, Close), but the second one contains several additional places (Done, Confl, Approve, 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.

Both GN models contain three types of tokens, which can split and merge, thus carrying the dynamic nature of the models:

- ε-token, standing for the editors (users), who access (open) a file. This token initially enters the net through place Open.
- δ-token, staying only in place DB, and representing the database of files. The separate instances of files are tokens, noted by φ and once they split from the δ-token, they may move around the net.
- ι-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 ν. Once they split from the ι-token, they may move around the net.

The logic of the GN models 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 (L or C), the transition's serial number and the predicate's serial number.

Obviously, the GN model of the LMU approach (Figure 4) is the simpler of both. It contains two transitions, nine places and the above described three types of tokens.

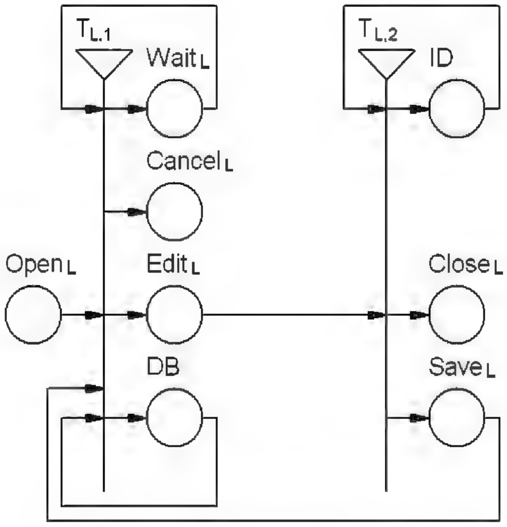


Figure 4: GN model of Lock – Modify – Unlock

Every ε-token enters the net via place $Open_L$ with an initial characteristic:

"Username (IP-address) of the editor. Requested file. Moment of access to file T_{Open}".

Token δ permanently stays in place DB with characteristic:

"List of files in the database with their Locked/Unlocked status and Current version ID".

Initially, all files in the database have status *Unlocked*, but during the model processing all files that are open for editing have their status changed to *Locked*. For example:

File	Status	Current version ID
$file_1$	unlocked	id_1
…	…	…
$file_k$	locked	id_k
…	…	…
$file_n$	unlocked	id_n

While token δ represents the whole database (repository), the separate instances of files from the database will be represented as φ-tokens.

Token ι 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 ι represents the generator of identifiers, the separate instances of revision identifiers will be represented as ν-tokens.

The first transition $T_{L,1}$ has four input and four output places

$$T_{L,1} =< \{Wait_L, Open_L, Save_L, DB\}, \{Wait_L, Cancel_L, Edit_L, DB\}, M_{L,1} >$$

related by means of the predicates in the following index matrix

$$M_{L,1} = \begin{array}{c|cccc} & Wait_L & Cancel_L & Edit_L & DB \\ \hline Wait_L & P_{L,1,3} & P_{L,1,4} & P_{L,1,1} & false \\ Open_L & P_{L,1,2} & false & P_{L,1,1} & false \\ Save_L & false & false & false & true \\ DB & P_{L,1,2} & false & P_{L,1,1} & true \end{array}.$$

where predicates $P_{L,1,1},\ldots,P_{L,1,4}$ have the following interpretations.

- $P_{L,1,1} =$ *"The status of the requested file by the current user is Unlocked."*
- $P_{L,1,2} = \neg P_{L,1,1} =$ *"The status of the requested file by the current user is Locked."*
- $P_{L,1,3} =$ *"The user is waiting for the file to get unlocked."*
- $P_{L,1,4} =$ *"The user cancels waiting for the file to get unlocked."*

When the truth value of predicate $P_{L,1,1}$ is true, token ε leaves place Open or Wait and enters place Edit. Simultaneously token δ in place DB splits to the same token δ and a new token φ, which represents a file from the database, requested by the current user for editing. Token φ enters place Edit with current characteristic:

"Name and contents of the requested file. Optional characteristics".

Examples of optional characteristics are some attributes of the latest file revision like identifier, author, timestamp, file size, revision summary, identifier of the preceding revision, etc., which are usually automatically attached to the current revision identifier.

Token δ makes one loop and returns in place DB and updates its characteristic by changing in the table the status of the file, corresponding to φ from Unlocked to Locked. It is noteworthy, that the status of the file is an important characteristic, which makes the difference between the $P_{L,1,1}$ and $P_{L,1,2}$ predicates.

So, when the ε-token from place $Open_L$ and the φ-token from place DB enter place $Edit_L$, they merge together under the name of the ε-token, which thus obtains the characteristic:

"Username. Filename. Moment T_{Lock} of locking the file. Status: Locked"

Predicate $P_{L,1,2}$ becomes true when an ε-token enters the net through place $Open_L$, requesting a file from the database, but the corresponding entry in the characteristic table of the δ-token in place DB reads that the requested file is Locked for editing. Hence, the δ-token remains in the DB without splitting another token, while the ε-token transfers to place $Wait_L$ with characteristic:

"Moment T_{Wait} when the editor has started waiting for the file to get unlocked."

Predicate $P_{L,1,3}$ is true as long as the editor is waiting for the file to get unlocked, i.e. the ε-token loops in place $Wait_L$ without obtaining any new characteristic.

Predicate $P_{L,1,4}$ becomes true when the editor decides to cancel waiting for the file to get unlocked, thus the ε-token from place $Wait_L$ transfers to place $Cancel_L$ with the new characteristic:

"Current moment of time T_{Cancel}. $Time_{Wait}$"

where $Time_{Wait}$ is the time spent in vain, in waiting for the file to get unlocked, and $Time_{Wait} = T_{Cancel} - T_{Wait}$.

Once that a token φ enters place $Save_L$, on the subsequent moment of GN functioning it will unconditionally transfer to place DB (hence the corresponding member of the index matrix $M_{L,1}$ is true, rather than a predicate) with a characteristic, which will be commented later.

The second transition $T_{L,2}$ has two input and three output places

$$T_{L,2} =< \{ID, Edit_L\}, \{ID, Close_L, Save_L\}, M_{L,2} >$$

related by means of the predicates in the following index matrix

$$M_{L,2} = \begin{array}{c|ccc} & ID & Close_L & Save_L \\ \hline ID & P_{L,2,2} & false & P_{L,2,2} \\ Edit_L & false & P_{L,2,1} & P_{L,2,2} \end{array}$$

where the predicates $P_{L,2,1}$ and $P_{L,2,s}$ have the following interpretations:

- $P_{L,2,1}$ = *"The user wants to finish one's work without saving the edit".*
- $P_{L,2,2}$ = *"The user wants to finish one's work and save the edit".*

When a token ε (merged, as described above) enters place Edit_L, it will stay in this place as long as the editor is working on the file. When the editor finishes working on the file, the token will leave place Edit_L because either predicate $\mathsf{P}_{L,2,1}$ or predicate $\mathsf{P}_{L,2,2}$ will become true, depending on the particular scenario: the user may either choose to close the file without saving or s/he may choose to save the edits made (as it usually happens). The first scenario will also cover some exceptional cases like power failure, resulting in unexpected system shut-down, etc.

In case of a real simulation of the model, the time for editing can be implemented by choosing a random value from a range of plausible durations of editing, and attributing it to token ε in place Edit_L as a second attribute of the characteristic. Thus, when this time interval elapses, as registered by the model's internal time measuring system, the precise time of the file being locked can be calculated.

When predicate $\mathsf{P}_{L,2,1}$ is true, i.e. the editor chooses to leave without saving the file, the ε-token will split to two tokens: the same token ε and token φ. The newly derived ε-token will transfer to place Close_L with the characteristic:

"Moment T_{Close} of closing the file. $Time_{Close}$"

where $Time_{Close} = T_{Close} - T_{Lock}$ is the duration of keeping the file locked, even though no changes have occurred as a result of the editing session.

The newly derived φ-token will enter place Save_L with characteristic:

"File content: Unchanged. Version identifier: Unchanged. File status: Unlocked".

When predicate $\mathsf{P}_{L,2,2}$ is true, i.e. the editor chooses to save the introduced changes, three actions will take place. First, the generator of version identifiers will issue the respective id-number that will be attributed to the newly saved file version, and will then increment with one (so that the subsequent revision of a file in the database obtains a different version identifier). For this sake, the ι-token (with current characteristic the natural number $ID_{Current}$), staying in place ID, will split to two tokens: the new token ν and the same token ι. Token ν will transfer to place Save_L with characteristic:

"Current value of token ι: $ID_{Current}$".

Token ι, as obtained after the split, will enter place ID with characteristic:

"$ID_{Current} + 1$".

On the other hand, the ε-token from place Edit_L will again split to two tokens: the same token ε and token φ (which corresponds to the previously noted token φ as derived from place DB). The newly obtained ε-token will transfer to place Close_L with the characteristic:

"Moment T_{Save} of saving the file. $Time_{Save}$"

where $Time_{Save} = T_{Save} - T_{Lock}$ is the duration of editing the file and keeping it locked for other user to edit in the meanwhile. (The difference between $Time_{C\text{-}lose}$ and $Time_{Save}$ is only in their semantics.)

The newly derived φ-token enters place Save_L where it merges with token v under the name φ and obtains characteristic:

"File content: Changed. Version identifier: ID_{Curren}*.*
File status: Unlocked".

Finally, we will return to the characteristic of φ-token obtained on its unconditional transfer from place Save_L back to place DB. Practically:

- it describes the fact of unlocking the file, no matter edited or not; and
- in case of saving the changes, it adds the new version to the file history and makes it the one, currently visible by default for all readers in the system.

So, the φ-token from place Save_L enters place DB where it merges with the host token δ with characteristics, depending on the particular characteristic values, carried by the φ-token on the previous step of the model functioning.

4 Conclusions

The present communication describes in terms of generalized nets one of the two major approaches for edit conflict resolution in version control systems, the so-called "Lock – Modify – Unlock". The second part of the research will be devoted to the other mechanism of this kind, called "Copy – Modify – Merge", which is a little bit more sophisticated. Finally, both approaches will be simulated with the forthcoming GN simulator and the results will be compared, thus presenting a decision support tool for developers and users of this kind of software applications..

Acknowledgements

This work is supported by the European Social Fund and Bulgarian Ministry of Education, Youth and Science under the Operative Program "Human Resource Development". Grant BG051PO001 – 3.3.04/40.

It is also partially supported by project "Research of Decision Making Models in Fuzzy Environment", 010077/2007.

Figures 1-3 are taken from the book "Version Control with Subversion" by Collins-Sussman B., 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] Collins-Sussman B., Fitzpatrick B. W., Pilato C. M. (2004), Version Control with Subversion, O'Reilly Media, http://svnbook.red-bean.com/.

The papers presented in this Volume 2 constitute a collection of contributions, both of a foundational and applied type, by both well-known experts and young researchers in various fields of broadly perceived intelligent systems.
It may be viewed as a result of fruitful discussions held during the Eighth International Workshop on Intuitionistic Fuzzy Sets and Generalized Nets (IWIFSGN-2009) organized in Warsaw on October 16, 2009 by the Systems Research Institute, Polish Academy of Sciences, in Warsaw, Poland, Centre for Biomedical Engineering, Bulgarian Academy of Sciences in Sofia, Bulgaria, and WIT – Warsaw School of Information Technology in Warsaw, Poland, and co-organized by: the Matej Bel University, Banska Bistrica, Slovakia, Universidad Publica de Navarra, Pamplona, Spain, Universidade de Tras-Os-Montes e Alto Douro, Vila Real, Portugal, and the University of Westminster, Harrow, UK:

http://www.ibspan.waw.pl/ifs2009

The Eighth International Workshop on Intuitionistic Fuzzy Sets and Generalized Nets (IWIFSGN-2009) has been meant to commence a new series of scientific events primarily focused on new developments in foundations and applications of intuitionistic fuzzy sets and generalized nets pioneered by Professor Krassimir T. Atanassov. Moreover, other topics related to broadly perceived representation and processing of uncertain and imprecise information and intelligent systems are discussed.

We hope that a collection of main contributions presented at the Workshop, completed with many papers by leading experts who have not been able to participate, will provide a source of much needed information on recent trends in the topics considered.

ISBN-13 9788389475305
ISBN 838947530-8
9 788389 475305