

# Efficient Graph Partitioning Algorithms for Collaborative Grid Workflow Developer Environments

Gergely Sipos<sup>1,2</sup> and Péter Kacsuk<sup>1</sup>

<sup>1</sup> MTA SZTAKI, Hungarian Academy of Sciences, Budapest, Hungary

<sup>2</sup> EGI.eu, Amsterdam, The Netherlands

{sipos, kacsuk}@sztaki.hu

**Abstract.** Collaborative editing systems allow a group of users to view and edit a shared item from geographically dispersed sites. Consistency maintenance in the face of concurrent accesses to shared entities is one of the core issues in the design of these systems. The paper introduces a lock based solution and three associated algorithms by which grid workflow developer environments can enable concurrent access to grid applications for multiple persons. The methods assure that collaborators cannot break the consistency criteria of workflows by introducing cycles or invalid edges to them. A formal analysis of the three algorithms is provided, focusing on the number of users that can simultaneously edit the same graph. Based on the findings an integrated algorithm is defined and it allows even more users to collaborate during workflow development.

**Keywords:** workflow, collaboration, real-time, groupware, locking, CSCW.

## 1 Introduction

Grid systems span over multiple administrative domains and interconnect heterogeneous resources to provide scalable infrastructures for data and compute intensive applications. Although grids were flagged right from the beginning as powerful platforms for collaborative work [19], most of the current grid applications and environments do not, or poorly exploit this feature. Defining workflows on top of grid middleware is the most common form of utilizing grids [14]. In 2005 we coined the term “collaborative grid workflow” for workflows that a group of users define and execute on the Grid together using a multi-user groupware environment [1]. Since then collaborative grid workflows became highly useful vehicles for inter-disciplinary research [9], urban planning [10], gene sequence analysis [11], engineering [12] and other domains and cases where intangible assets and tangible resources of several organizations and individuals must be integrated in an efficient way.

Recent works from this field studied the concurrent access to grid workflows [6][11], but did not deal with inconsistency, a common issue in computer supported collaborative work (CSCW) environments. Our previous papers [13][20] proposed lock based concurrency control and graph partitioning algorithms to protect the following three consistency criteria of workflows during collaborative editing sessions: (1) workflows must remain acyclic; (2) edges in the workflow must point to existing

vertexes; (3) maximum one input edge can refer to an input parameter of a workflow node (maximum one edge can provide data for an input channel of a grid service/job). Those papers introduced three different graph partitioning algorithms that can allocate sub-graphs to workflow editors in such a way, that the modifications performed on these sub-graphs cannot break the consistency rules of the complete graph. We proved that the algorithms guarantee workflow consistency without dropping any user's changes or forcing any user to compensate his/her committed modifications. Maintaining consistency of workflows is highly important in any CSCW application because faults not discovered at early stages can be debugged later only at high costs [18].

In the current paper we evaluate and compare these three graph partitioning algorithms and suggest an improved solution based on the findings. We use a user-centric metric for the evaluation: *the number of persons that an algorithm allows to share and edit a single graph simultaneously*. The contribution of the paper to the parallel computing field is a lock based solution and associated algorithm to partition workflow graphs among multiple parties. We contribute to the CSCW domain with a theoretical evaluation of consistency-aware groupware protocols.

The next section provides an overview of related works. In Section 3 the concept of lock based collaborative editing of grid workflows is described. The problems related to workflow consistency are explained in Section 4, where three consistency-aware graph partitioning protocols are introduced. Section 5 introduces the metric we use to evaluate lock partitioning algorithms from the users' point of view. It also gives a comparative analysis of the three algorithms based on this metric and introduces a more advanced, 4<sup>th</sup> algorithm. We show that this new algorithm allows even more developers to share a graph, and can contribute to the quicker completion of workflows. In Section 6 additional properties of the solutions are discussed, such as fairness, deadlock and starvation, and an overview of our prototype implementation of the system based on P-GRADE Grid Portal is also given. Our summary and conclusions are drawn in Section 7.

## 2 Related Work

Real-time collaboration through shared, editable entities has recently become an intensively researched topic within CSCW. Most of the related efforts focused on enabling concurrent editing of items that are either unstructured (such as a drawing) or have unpredictable structure (such as a document). While unstructured items do not have consistency requirements and do not require consistency maintenance frameworks, freeform items demand so complicated and unpredictable consistency maintenance protocols that these are too difficult to develop.

Usage of groupware approaches in grid computing is a less researched field. Although there are some grid environments that support the collaboration of users, these systems are often tightly coupled with one particular application (so users do not have flexibility on what information, application they would like to share with each other) e.g. [3][4], or they do not support real-time collaboration. For example, MyExperiment.org is a Web 2 style community site where e-scientist can publish, discover and download workflows [5], but it does not include any support for concurrent editing of these entities thus it is not concerned of workflow structures and inconsistencies.

Some recent efforts in grid workflow research have focused on the knowledge sharing aspects of collaborative workflows in multi-organizational environments. While these works provide detailed analysis on the users' awareness and propose system architectural components, they rarely discuss concurrency control mechanisms and implementations [8][12]. There are a few exceptions though. In our first paper on the topic we defined a lock based concurrency control framework using P-GRADE Grid Portal [1][2]. This solution does not protect the consistency of workflows. The HOBBS environment also uses locks for concurrency control during workflow development [11]. However, HOBBS does not guarantee the consistency of shared workflows, the owner of the graph must compensate the collaborators' work if the team results inconsistent, or inefficient graph structure.

Version Control Systems (VCS) such as CVS, SVN, Git, Mercurial are the typical tools to support the concurrent engineering of software. However these environments cannot help in the collaborative development of workflows because (i) workflows are typically stored in a single file (e.g. workflow description file) meanwhile VCS supports collaboration on file sets; and (ii) VCS work at the level of text and cannot interpret and resolve conflicts that happen at a higher abstraction level (in our case at the level of graph vertexes or edges).

The most widely adopted method to evaluate groupware environments is the analysis through real or artificial user scenarios [15]. At the same time this method is also criticized, mainly because the whole spectrum of groupware use cases cannot be covered in this way [16]. Our work focuses on a narrow aspect of groupware usability, on the number of people that are allowed by a graph partitioning algorithm to work concurrently on a single workflow. Because of the well defined focus, we can perform a purely theoretical analysis. Baeza-Yates and Pino analyze groupware systems in a similar fashion [17]. However, they focus on the quality of shared entities, and try to describe this parameter as the function of group size.

### 3 Collaborative Editing of Grid Workflow Graphs

The collaborative development of workflows requires concurrent access to a single graph by multiple users. Collaborative development tools (groupware tools) must assure that the users' contributions are integrated into a single, coherent application without resulting loss of data or invalid state. Grid workflows are developed manually, i.e. changes made on a workflow are results of human actions. In a collaborative environment if user *A*'s changes on a workflow is lost, or dropped due to a user *B*'s concurrent changes, then this results wasted, sometimes irreproducible effort for user *A*. As we described in our previous papers, *an important requirement for collaborative development of grid workflows is that no user's development session should be aborted because of another user's concurrent work* [13][20].

Turn-taking, serialization (often extended with operation transformation) and locking are the most commonly used concurrency-control techniques in groupware environments [12]. In our earlier works we argued that lock based concurrency control matches most with the needs of workflow developers, particularly of grid workflow developers [1][6]. We suggest locks at graph component level, i.e. vertexes and edges in a workflow are the lockable units. In our collaborative workflow editor

architecture the lock manager appears as a central component. The role of the lock manager is to accept lock requests from the users, evaluate these requests and grant or deny the locks based on compatibility rules and protocols (See also Fig 3.).

In our previous papers we introduced three different algorithms that can be integrated into a lock manager to grant and deny locks [13][20]. We call these algorithms as “*lock evaluator algorithms*”. A lock evaluator algorithm can be imagined as a function that receives a workflow graph and a set of lock request as inputs, and gives a partitioning of the graph as output. The lock requests are generated by the workflow developers, and each of the requests refers to a subset of the vertexes and subset of the edges in the workflow. The person who generated the request (through his/her editor) would like to modify these components.

A lock evaluator algorithm partitions the graph to locked and unlocked parts. A locked sub-graph minimally contains those elements of the workflow that a given lock requestors’ editing work will affect and must be locked for him/her to avoid conflicts. If a lock cannot be granted, then the user must be informed about the denial so he/she can modify the request or submit it at a later time. The way how this can be done is outside of the focus of this paper.

When multiple users work on the same graph concurrently, then a series of locking requests reach the lock evaluator algorithm. The algorithm evaluates each request separately, and allocates different parts of the graph for the users:

$$G, R^S \rightarrow \text{Lock evaluator algorithm} \rightarrow G = G_1 \cup G_2 \cup \dots \cup G_n \cup G^U$$

where  $G$  marks the workflow graph on which  $n$  users work concurrently.  $R^S = R_1, R_2, \dots, R_n$  marks the lock requests of the users.  $R_i$  includes those graph components (vertices, edges) that user  $i$  wants to modify on the graph. Such a request can be generated by the user e.g. through the graphical editor where he/she selects the workflow components that he/she wishes to modify.  $G_1, G_2, \dots, G_n$  mark  $n$  sub-graphs of  $G$  that become locked for the users.  $G_i$  gets locked for the owner of the  $R_i$  request.  $G^U$  is the sub-graph of  $G$  that remains unlocked even after all the  $n$  users begin to concurrently edit the graph. Note that each of the sub-graphs contains some vertices and some edges from  $G$ , but a  $G_i$  sub-graph is not necessarily connected. If e.g. a user intends to modify the first and the last vertexes of a pipe-line graph, then his/her locked sub-graph may contain only these two vertexes so the sub-graph is disconnected. Because of using exclusive locks  $G_i \cap G_j = \emptyset, \forall i, j, 1 \leq i, j \leq n, i \neq j$  and  $G_i \cap G^U = \emptyset, \forall 1 \leq i \leq n$ .

## 4 Consistency of Grid Workflow Graphs

Collaborative workflow editors must assure that editing sessions bring graphs from one consistent state to another consistent state. Consistency of a workflow can be defined at two levels: (1) component-level consistency: consistency of individual vertexes, individual edges; (2) graph-level consistency: consistency of multiple, connected components.

Component-level consistency defines what a valid parameter set is for a given workflow vertex or edge (e.g. what parameters can describe a Grid service or Grid job

in the workflow). *In the workflow domain that we work in the following three graph-level consistency criteria were found for workflow graphs [20]:*

1. There can be no “dangling” edge in the graph, i.e. an edge that refers to non-existing (already deleted) vertex as its source and/or sink.
2. Maximum one edge can provide input data for an input channel of a vertex. If multiple edges are connected to the same input channel, then the execution of the workflow would be unpredictable as the data items that arrive through the edges overwrite each other. (This condition still allows a vertex to have more than one input channels.)
3. The graph must be acyclic.

Our research focuses on the graph-level consistency rules because in a multi-user editing scenario no party has the complete view of the whole workflow, thus maintaining graph-level consistency is not self-evident. In [20] we defined a protocol that – if implemented by a lock evaluator algorithm – can prevent dangling edges and multiple incoming edges. The protocol basically prohibits users to hold lock on an edge without holding locks on the source and the sink vertexes of this edge. The protocol automatically extends a user’s locked sub-graph with those vertexes that are connected to the user’s edges.

It has been also shown in [20] that not a single, but several different protocols exist that can prevent cycles in workflows. These protocols partition workflow graphs in such a way that no edge creation operations within these sub-graphs can result in a cycle in the complete graph. In [20] we defined two of such algorithms, and in [13] we gave a third algorithm. In the current paper our goal is to evaluate the usability of these algorithms, so we shortly summarize how the three algorithms work.

The three algorithms implement the above described protocol so they protect against invalid edges by extending the users’ locked sub-graphs with extra vertexes. The algorithms do not allow any component (any vertex or any edge) to be locked by two users at the same time. Should the same component be locked by two concurrent users the algorithms return a lock denial answer for the second user. (The one whose lock request reached the system later in time.)

The first algorithm (to be called v1 algorithm now) locks complete branches of a graph, i.e. if a  $V$  vertex needs to be locked for user  $A$ , then every child vertex of  $V$  and every child edge of  $V$  becomes locked for  $A$  too. If either  $V$ , or any of its child components are already locked for a user  $B$ , then  $A$ ’s locking request is denied, the system does not identify a smaller, but lockable sub-graph.

The second algorithm (to be called v2 algorithm now) is an improved version of v1, because it uses two types of locks. Those components that the user  $A$  requested to lock become locked with USER lock (U lock). The child vertexes and child edges of these U-locked components become locked with SYSTEM lock (S lock). While U locks are visible for users and mark editable components, S locks are invisible to users and are used only by the lock evaluator algorithm to decide about lock compatibility. U locks are incompatible with each other, i.e. no more than one user can have U lock on a component. On the other hand, we proved that multiple S locks can exist on the same component, moreover, in some cases both S lock and U lock is allowed on the same component [20]. It has been also proven that *a lock request must be*

*denied only, if it puts U lock on a component that already has S lock and at the same time it puts S lock on a component that already has U lock.*

In [13] we defined a third algorithm (to be called v3 algorithm now). V3 uses only one lock (like the v1 algorithm, or like to U lock in the v2 algorithm), and it applies a different concept, called “contiguity graph” to decide about granting/denying locks. A *contiguity graph is a graph that represents the connections among locked sub-graphs, and among locked sub-graphs and unlocked components of a workflow.* In the contiguity graph of a G workflow graph every locked sub-graph of G appears as a single node; every edge that connects two locked sub-graphs together appears as an edge, and every unlocked component (vertex or edge) appears as it is (a vertex or an edge). When a new lock request arrives from a user the v3 algorithm generates the contiguity graph of the workflow and denies the lock only if it generates a cycle in the contiguity graph, i.e. if the vertex that represents the newly locked sub-graph closes a cycle in the contiguity graph.

## 5 Collaborative Performance of Lock Algorithms

The three lock evaluator algorithms (v1, v2 and v3) provide such partitioning of acyclic graphs that the consistency rules cannot be broken by any of the collaborative users. *The three algorithms use different policies to evaluate lock requests and can result different partitioning of a graph. In the same situation one algorithm can deny the lock request, while another algorithm can grant the locks. From the users’ point of view the algorithm that allows the most developers to work on the same graph concurrently provides the best collaborative performance.* The more users can access the workflow at the same time the sooner they can finish the graph definition process and can proceed to workflow execution. Because the three algorithms protect the consistency of graphs, there is no need to include a consistency checking stage after the workflow definition process. Workflow execution can begin right away.

The lock grant/denial decision of a lock evaluator algorithm depends *only* on the current state of the workflow graph, the topology of the existing locks, and the topology of the requested locks. The decision is independent from several other factors, such as the unsaved parts of the graph (these are visible only in the client side editors), the start time and finish time of an editing session, etc. When an algorithm evaluates the  $R^S = R_1, R_2, \dots, R_n$  sequence of lock requests on a G graph, then, besides partitioning G to locked sub-graphs and an unlocked sub-graph, *the algorithm implicitly also returns a 0 or a 1 value for each lock requestor.* 0 means that the lock request is denied, 1 means that the locks are granted. *The more 1 digits are given by an algorithm for an  $R^S$  lock request on a G graph the more users are allowed to concurrently edit the workflow. Consequently, by comparing the binary vectors of the algorithms in a given editing situation, the collaborative performance of these algorithms can be compared.*

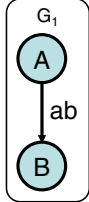
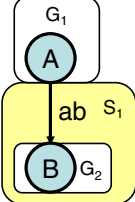
**Definition 1.** Given a G graph and an  $R^S = R_1, R_2, \dots, R_n$  sequence of lock requests. We say that an X lock evaluator algorithm *has better collaborative performance* than an Y algorithm, if there are more 1 digits in the vector returned by X than in the vector returned by Y.

**Definition 2.** We say that an X lock evaluator algorithm has better *overall collaborative performance* than an Y algorithm, if for *any* G graph and  $R^S = R_1, R_2, \dots, R_n$  sequence of lock requests there are at least as many 1 digits in the vector returned by X than in the vector returned by Y, and there is at least one F graph with one  $S^S = S_1, S_2, \dots, S_n$  lock requests sequence where more 1 digits are in the vector returned by X than in the vector returned by Y.

**5.1 Comparing the v1 and v2 Algorithms**

It is easy to see that the v2 algorithm provides a better *overall collaborative performance* than the v1 algorithm. In situations when a user’s lock request does not require S lock on the graph (the requested components do not have child nodes) the v2 algorithm results exactly the same lock topology than the v1 algorithm. In these cases the two algorithms allow the same persons to work on the graph and they return the same binary vectors.

However, when S locks must be used by v2 and only S locks collide during the evaluation of the request, then v2 allows more people to edit the workflow. See a simple example for this in Fig 1. *As a consequence, the v2 algorithm gives better user satisfaction than the v1 algorithm in any collaborative workflow editor environment.*

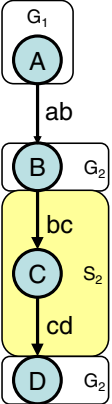
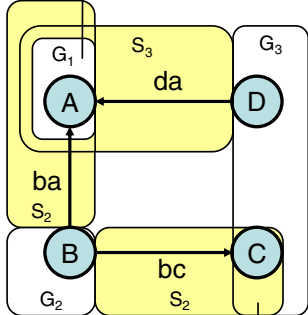
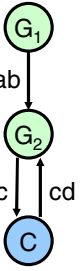
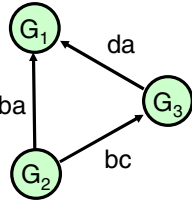
Lock requests	Locks issued by v1	Locks issued by v2
$R_1 = \{A\}$ $R_2 = \{B\}$	 <p><math>R_2</math> causes lock collision with <math>G_1</math>, the <math>R_2</math> request is denied.</p>	 <p>S lock is compatible with U lock, both <math>R_1</math> and <math>R_2</math> are served.</p>

**Fig. 1.** Collaborative workflow development scenario to demonstrate that the v2 algorithm provides a better collaborative performance than v1. (Notations:  $G_1, G_2$ : sub-graphs locked with U lock for  $R_1$  and  $R_2$ ;  $S_2$ : sub-graph locked with S lock for  $R_2$ ).

**5.2 Comparing the v2 and v3 Algorithms**

The situation with the v2 and v3 algorithms is more complicated. We found, that in some editing scenarios the v2 algorithm allows more users to work on the graph, while in other cases the v3 algorithm provides better collaborative performance. Fig 2. gives examples for each of these situations.

The second column of Fig 2. shows two users requesting locks on a G graph that consists of 4 vertexes and 3 edges. While v2 allows both users to work with the requested components, v3 grants locks only for the first person because the second user’s request results cycle in the contiguity graph. Consequently, v2 provides better collaborative performance in this case.

	v2 provides better performance than v3	V3 provides better performance than v2
Lock requests	$R_1=\{A\}$ $R_2=\{B, D\}$	$R_1=\{A\}$ , $R_2=\{B\}$ , $R_3=\{C, D\}$
Locks issued by v2		<p>Putting an S lock on an existing U lock</p>  <p>Putting an U lock on an existing S lock</p>
ontiguity graph of v3		
Evaluation outcome	<ul style="list-style-type: none"> <li>• <b>v2 allows</b> lock for <math>R_2</math> because it causes no lock collision. V2 returns the [1, 1] binary vector.</li> <li>• <b>v3 denies</b> lock for <math>R_2</math> because there is cycle in the contiguity graph. V3 returns the [1, 0] binary vector.</li> </ul>	<ul style="list-style-type: none"> <li>• <b>v2 denies</b> lock for <math>R_3</math> because it puts S lock on U lock and put U lock on S lock. V2 returns the [1,1,0] binary vector.</li> <li>• <b>v3 allows</b> lock for <math>R_3</math> because there is no cycle in the contiguity graph. V3 returns the [1,1,1] binary vector.</li> </ul>

**Fig. 2.** Collaborative workflow development scenarios that demonstrate the difference in the collaborative performance of the v2 and v3 lock evaluator algorithms. (Notations:  $G_1$ : sub-graph locked with U lock for  $R_1$ ;  $S_1$ : sub-graph locked with S lock for  $R_1$ , ...).

The third column shows 3 users requesting locks on a graph (on a different graph than in the second column). In this case v2 denies the lock for the third user, because the request puts U lock on S lock and puts S lock on U lock *at the same time*. V3 grants locks for all the three users, and provides better collaborative performance in this case.

These examples demonstrate that depending on the topology of the workflow graph and the topology and order of the lock requests, sometimes the v2, sometimes



the v3 algorithm allows more users to work on a graph. Because actions cannot be predicted in a generic groupware environment we cannot know in advance what graph and in what distribution will be developed. In order to be prepared for any editing situation we should define a new, v4 lock evaluator algorithm that provides better overall collaborative performance than v2 and v3. (We should not consider v1 any longer because v2 provides better overall performance.)

The number of possible editing cases in a collaborative workflow developer environment is huge. Any directed acyclic graph can be extended in numerous ways and going through these cases one by one is impossible. We instead propose a different and relatively simple solution to achieve v4: v4 can be created with the integration of the v2 and v3 algorithms. This integrated v4 algorithm first generates the S locks as it is done by v2 and checks if the requested locks can be granted. If they can, then the algorithm allocates the locks, otherwise generates the contiguity graph as it is done by v3. If the contiguity graph allows the locks (i.e. there is no cycle in it) then the locks are granted. The v4 algorithm allows the locks if either v2 or v3 allows it. V4 denies the lock only if both v2 and v3 denies. Because both v2 and v3 prevents workflow inconsistencies, either of them allow the editing work of a user the final workflow will not break the consistency rules.

## 6 Discussion and Implementation

All the four lock evaluator algorithms (v1, v2, v3 and v4) provide such a partitioning of acyclic graphs that the three consistency rules cannot be broken by any editor. However, none of the solutions guarantee deadlock free editing. It can happen that user *A* locks sub-graph  $G_a$ , user *B* locks sub-graph  $G_b$  and they both wait for each others sub-graphs to become unlocked in order to extend their own sub-graphs with those components. Prohibiting users to hold locks on more than one workflow component could be a solution as it is applied in the GroupGraph system [23]. However, in this way operations that affect multiple components (e.g. a definition of a new edge requires two end vertexes) cannot be performed. That is why we suggest instead the inclusion of awareness tools in the editors. These tools could show the topology of locks on the whole workflow, can help users contact each other to discuss and manually resolve deadlock situations. Groupware widgets, such as the MAUI Toolkit [7] can be used to increase the developers' group awareness.

To demonstrate the introduced concepts we extended a single-user grid workflow environment with collaborative capabilities. We choose P-GRADE Grid Portal [21] for this purpose, because it is a widely used tool in production grids (e.g. in the EGEE grid, UK NGS, US OSG) and because it uses a client-server architecture. P-GRADE Portal has a Web based graphical server for the execution and monitoring of data-driven computational workflows, and it has a client side editor for the editing of graphs. One can define a grid workflow in this editor by connecting sequential and parallel batch jobs into data driven applications. P-GRADE Portal server uses Condor DAGMan workflow engine to schedule workflows to computational resources, so the acyclic property of graphs and the proper connection between vertices are important consistency requirements in the system. We extended both the client and the server of P-GRADE Grid Portal with collaborative features. The new architecture is presented in Fig 3.

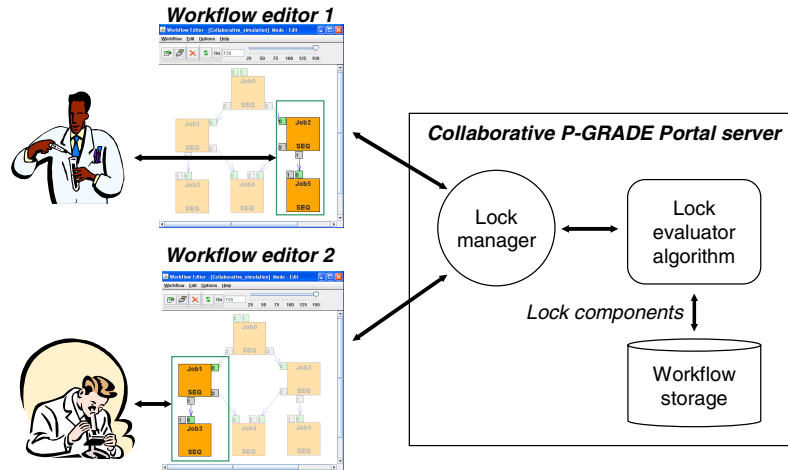


Fig. 3. Architectural overview of Collaborative P-GRADE Grid Portal

Users can request locks on components of a workflow in the collaborative editor. Requests are forwarded by the editors to the server side lock manager then to the lock evaluator algorithm to estimate which components of the graph need to be, and can be locked. Multiple users can possess locks on the same graph simultaneously. Fig 3. demonstrates this with showing different users' sub-graphs within green boxes, while keeping the rest of the graph semi-transparent. Users can share their graph modifications with each other through the server in real-time, and if they wish can see each others modifications as well. Because the locked components can be modified locally, the system provides good responsiveness. P-GRADE Grid Portal uses the server side file system to store workflows and locks, however any other storage could be used as long as workflow components can be flagged „locked” and „unlocked” in it.

Our current implementation does not maintain any waiting queue for denied lock requests, thus users with denied requests must manually re-request locks once they see that other users finished working with the components they need.

## 7 Conclusions

The paper provided an evaluation of three algorithms that can be used for the intelligent portioning of workflow graphs in real-time collaborative editing systems, and made suggestion for an integrated solution. *The locked based collaborative development concept assures that no user's editing session is aborted, no user's effort is wasted, and that the consistency of workflows is guaranteed.* The analysis of the three algorithms revealed that the second algorithm performs better than the first, but one cannot decide for the second or the third algorithm so easily. We showed a simple solution to achieve an overall best algorithm with the integration of this second and third solution.

The lock based concurrency control groupware mechanisms has been implemented in the prototype version of the Collaborative P-GRADE Grid Portal and it will be

used by the Grid Application Support Centre (GASuC) [22] of MTA SZTAKI to enable scientific applications on the European Grid Infrastructure (EGI). EGI is based on the federation of individual National Grid Infrastructures coordinated by the EGI.eu organization. The extension of an existing single-user grid workflow tool demonstrates that an originally single-user grid software can be transformed to a groupware application. Although our tool has been defined for grid application developers, the graph partitioning algorithms are independent from grid computing and could be used in other workflow based groupware tools as well.

## References

1. Sipos, G., Lewis, G.J., Kacsuk, P., Alexandrov, V.N.: Workflow-oriented Collaborative Grid Portals. In: Sloot, P.M.A., Hoekstra, A.G., Priol, T., Reinefeld, A., Bubak, M. (eds.) EGC 2005. LNCS, vol. 3470, pp. 434–443. Springer, Heidelberg (2005)
2. Kacsuk, P., Sipos, G.: Multi-Grid, Multi-User Workflows in the P-GRADE Portal. *Journal of Grid Computing* 3(3-4), 221–238 (2005)
3. Nacar, A.M., et al.: VLab: collaborative Grid services and portals to support computational material science. *Concurrency and Computation: Practice and Experience* 19(12), 1717–1728 (2007)
4. Yu, O., Lia, A., Caoa, Y., Yina, L., Liaoa, M., Xua, H.: Multi-domain Lambda Grid data portal for collaborative Grid applications. *Future Generation Computer Systems* 22(8), 993–1003 (2006)
5. Goble, C.A., De Roure, D.C.: MyExperiment: Social Networking for Workflow-Using E-scientists. In: *Proceedings of the 2nd Workshop on Workflows in Support of Large-scale Science*, Monterey, California, USA. ACM, New York (2007)
6. Sipos, G., Kacsuk, P.: Collaborative Workflow Editing in the P-GRADE Portal. In: *Proc. of Microcad 2005 International Conference*, Miskolc, Hungary, pp. 353–358 (2005)
7. Hill, J., Gutwin, C.: The MAUI Toolkit: Groupware Widgets for Group Awareness. *Journal of Computer Supported Cooperative Work (CSCW)* 13(5-6), 539–571 (2004)
8. Schuster, H., Baker, D., Cichocki, A., Georgakopoulos, D., Rusinkiewicz, M.: The collaboration management infrastructure. In: *Proc. of ICDE Conference*, San Diego, California, USA, pp. 677–678 (2000)
9. Zhao, Z., Booms, S., Belloum, A., de Laat, C., Hertzberger, B.: VLE-WFBus: A Scientific Workflow Bus for Multi e-Science Domains. In: *Proc. of Second IEEE International Conference on E-Science and Grid Computing (e-Science 2006)*, Amsterdam, The Netherlands, p. 11 (2006)
10. Müller, J.M., Zhang, G., Lapayre, J.C., Müller, P.: Service-oriented Support of Cooperative Workflows. Considerations for Urban Planning Processes. In: *Proc. of 11th Conference of the Association for Information And Management*, Luxemburg (2006)
11. Held, M., Blochinger, W.: Structured Collaborative Workflow Design. *Future Generation Computer Systems* 25(6), 638–653 (2009)
12. Friese, T., Smith, M., Freisleben, B., Reichwald, J., Barth, T., Grauer, M.: Collaborative Grid Process Creation Support in an Engineering Domain. In: Robert, Y., Parashar, M., Badrinath, R., Prasanna, V.K. (eds.) *HiPC 2006*. LNCS, vol. 4297, pp. 263–276. Springer, Heidelberg (2006)
13. Sipos, G., Kacsuk, P.: Efficient Partitioning of Graphs in Collaborative Workflow Editor Systems. In: *Proceedings of IADIS International Conference Collaborative Technologies*, Freiburg, Germany (to appear 2010)

14. McPhillipsa, T., Bowersa, S., Zinn, D., Ludäscher, B.: Scientific workflow design for mere mortals. *Future Generation Computer Systems* 25(5), 541–551 (2008)
15. Pinelle, D., Gutwin, C.: A Review of Groupware Evaluations. In: *Proceedings of Ninth IEEE WETICE 2000 Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, Gaithersburg, Maryland, pp. 86–91 (2000)
16. Araujo, R.M., Santoro, F.M., Borges, M.R.S.: The CSCW Lab for groupware evaluation. In: Haake, J.M., Pino, J.A. (eds.) *CRIWG 2002. LNCS*, vol. 2440, pp. 222–231. Springer, Heidelberg (2002)
17. Baeza-Yates, R., Pino, J.: A first step to formally evaluate collaborative work. In: *Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work: the Integration Challenge*, Phoenix, Arizona, USA, November 16–19, pp. 56–60 (1997)
18. Dewan, P., Riedl, J.: Toward computer-supported concurrent software engineering. *IEEE Computer* 26(1), 17–27 (1993)
19. Foster, I., Kesselman, C.: Computational Grids. In: Foster, I., Kesselmann, C. (eds.) *The Grid: Blueprint for a New Computing Infrastructure*, pp. 2–48. Morgan Kaufmann, San Francisco (1999)
20. Sipos, G., Kacsuk, P.: Maintaining Consistency Properties of Grid Workflows in Collaborative Editing Systems. In: *Proc. of Grid and Collaborative Computing Conference (GCC 2009)*, pp. 168–175. IEEE-Publishing, Lanzhou (2009)
21. P-GRADE Grid Portal,  
<http://www.portal.p-grade.hu>,  
<http://sourceforge.net/projects/pgportal/>
22. MTA SZTAKI Grid Application Support Centre (GASuC),  
<http://www.lpds.sztaki.hu/gasuc>
23. Lima Filho, H.A.S., Hirata, C.M.: GroupGraph: A Collaborative Hierarchical Graph Editor Based on the Internet. In: *Proceedings of the 35th Annual Simulation Symposium* (2002)