



Localising temporal constraints in scientific workflows

Jinjun Chen, Yun Yang*

CS3 – Centre for Complex Software Systems and Services, Faculty of Information and Communication Technologies, Swinburne University of Technology, PO Box 218, Hawthorn, Melbourne, Australia 3122

ARTICLE INFO

Article history:

Received 20 March 2009

Received in revised form 20 August 2009

Available online 22 November 2009

Keywords:

Scientific workflow

Temporal constraint

Localisation

ABSTRACT

Temporal constraints are often set when complex e-science processes are modelled as scientific workflow specifications. However, many existing processes such as climate modelling often have only a few coarse-grained temporal constraints globally. This is not sufficient to control overall temporal correctness as we can not find temporal violations locally in time for handling. Local handling affects fewer workflow activities, hence more cost effective than global handling with coarse-grained temporal constraints. Therefore, in this paper, we systematically investigate how to localise a group of fine-grained temporal constraints so that temporal violations can be indentified locally for better handling cost effectiveness. The corresponding algorithms are developed. The quantitative evaluation demonstrates that with local fine-grained temporal constraints, we can improve handling cost effectiveness significantly than only with coarse-grained ones.

© 2009 Elsevier Inc. All rights reserved.

1. Introduction and motivation

Scientific workflows often sit in sophisticated scientific applications such as climate modelling and astrophysics simulation [2,3,11,20,23]. They enable complex scientific computation to be performed step by step [10,12,19,21,30].

In reality, scientific workflows are normally time constrained as temporal correctness is critical to ensure the usefulness of execution results [4,8,13,22]. Consequently, temporal constraints are often set. The types of temporal constraints mainly include: upper bound, lower bound and fixed-time [8,13]. An upper bound constraint between two activities is a relative time value so that the duration between them must be less than or equal to it. A lower bound constraint between two activities is a relative time value so that the duration between them must be greater than or equal to it. A fixed-time constraint at an activity is an absolute time value such as 6:00pm by which the activity must be completed.

Comparing the three types of temporal constraints, we can find that conceptually a lower bound constraint is symmetrical to an upper bound constraint while a fixed-time constraint is a special case of upper bound constraint. The reasons are as follows. For a lower bound constraint, we often check whether the duration between its start and end activities is greater than or equal to (\geq) its value while for an upper bound constraint, we often check whether the duration between its start and end activities is less than or equal to (\leq) its value. Therefore, they are symmetrical to each other. As for a fixed-time constraint, the first activity of a scientific workflow is actually its start activity. Hence, a fixed-time constraint can be viewed as a special upper bound constraint whose start activity is the first activity and whose end activity is the one at which the fixed-time constraint is. Nevertheless, an upper bound constraint is conceptually more general than a fixed-time constraint as its start activity can be an intermediate activity rather than the first activity. Besides, different upper bound constraints can have different start activities while all fixed-time constraints have the same one which is the first activity.

* Corresponding author.

E-mail addresses: jchen@swin.edu.au (J. Chen), yyang@swin.edu.au (Y. Yang).

As such, in this paper, we focus on upper bound constraints only. The corresponding discussion and results can be symmetrically applied to lower bound constraints and adaptively simplified for fixed-time constraints.

In many scientific workflows such as climate modelling, we often have only a few coarse-grained upper bound constraints globally [2,24]. From the perspective of user needs, only a few coarse-grained upper bound constraints are intuitive and simple. However, from the perspective of specific scientific workflow execution, we cannot identify temporal violations locally in time for handling. Local handling affects fewer workflow activities than global handling with coarse-grained constraints, hence more cost effective. Therefore, we must investigate how to localise a group of fine-grained upper bound constraints based on coarse-grained ones so that we can identify temporal violations locally for better handling cost effectiveness. The existing related work has presented some background for the temporal aspect in scientific workflows. [4,24] analyses QoS (Quality of Service) including temporal QoS in scientific workflows on grid and discusses how to provide QoS including time. [14] examines key challenges in scientific workflow area including time aspect. [5] investigates multiple temporal consistency states in scientific/grid workflows. [16] discusses fault tolerance and recovery in scientific workflows including time management. [18] proposes a reference architecture for scientific workflow management in service computing environment. [25] analyses the overhead of scientific workflow execution in grid environment. [29] proposes a p2p based scientific workflow management architecture with time management included. [22] presents a method for dynamic verification of temporal constraints. [31] proposes a taxonomy for scientific workflow management. Several metrics are proposed to categorise scientific workflow management with time as one of them. [6–8] propose several strategies for selecting checkpoints for verifying temporal constraints.

However, the above existing work does not pay sufficient attention to how to localise fine-grained upper bound constraints. Hence, in this paper, we make an effort to fill this gap by systematically investigating the issue. We take one of coarse-grained upper bound constraints as the example to discuss how to localise fine-grained upper bound constraints within its timeframe. The corresponding results can be equally applied to each of other coarse-grained upper bound constraints. Based on the investigation, we develop the corresponding algorithms. With fine-grained upper bound constraints, we can achieve better cost effectiveness significantly than only with coarse-grained ones. The quantitative evaluation further demonstrates this result.

The paper is organised as follows. In Section 2, we summarise some time attributes of scientific workflows. In Section 3, we discuss how to localise fine-grained upper bound constraints including assignment and adjustment of them. The corresponding algorithms are developed. In Section 4, we conduct a quantitative evaluation which demonstrates that based on these algorithms we can achieve better handling cost effectiveness significantly than only based on coarse-grained upper bound constraints. Finally in Section 5, we conclude our contributions and point out future work.

2. Overview of timed scientific workflow representation

According to [1,17,27], based on the directed graph concept, a scientific workflow can be represented by a scientific workflow graph, where nodes correspond to activities and edges correspond to dependencies between them. To represent time attributes in a scientific workflow, we borrow some concepts from [13,22] such as maximum, mean or minimum duration as a basis. We denote the i th activity of a scientific workflow as a_i and its maximum duration, mean duration, minimum duration, run-time start time and run-time completion duration as $D(a_i)$, $M(a_i)$, $d(a_i)$, $S(a_i)$, $E(a_i)$ and $R(a_i)$, respectively. $M(a_i)$ means that statistically a_i can be completed around its mean duration. Other time attributes are self-explanatory. According to [9,28], $D(a_i)$, $M(a_i)$ and $d(a_i)$ can be obtained based on the past execution history which covers the delay time incurred at a_i such as setup delay, queuing delay, synchronisation delay, network latency and so on. The detailed discussion on how to obtain and set $D(a_i)$, $M(a_i)$ and $d(a_i)$ is outside the scope of this paper and can be found in [9,28]. For a specific execution of a_i , the delay time is included in $R(a_i)$. Normally, we have $d(a_i) \leq M(a_i) \leq D(a_i)$ and $d(a_i) \leq R(a_i) \leq D(a_i)$.

If there is a path from a_i to a_j ($i \leq j$), we denote the maximum duration, minimum duration, mean duration, run-time real completion duration between them as $D(a_i, a_j)$, $d(a_i, a_j)$, $M(a_i, a_j)$ and $R(a_i, a_j)$, respectively [9,28]. If there is an upper bound constraint between a_i and a_j , we denote it as $U(a_i, a_j)$ and its value as $u(a_i, a_j)$. For convenience, we only consider one execution path in the scientific workflow without losing generality. As to a selective or parallel structure, for each branch, it is an execution path. For an iterative structure, from the start to the end, it is still an execution path. Therefore, for the selective/parallel/iterative structures, we can also apply the results achieved from one execution path.

Besides the above time attributes, four temporal consistency states have been identified and defined in [5,8] which are SC (Strong Consistency), WC (Weak Consistency), WI (Weak Inconsistency) and SI (Strong Inconsistency). We summarise their definitions in Definitions 1, 2 and 3. The detailed discussion about the four consistency states can be found in [5,8].

Definition 1. At build-time stage, $U(a_i, a_j)$ is said to be of SC if $D(a_i, a_j) \leq u(a_i, a_j)$, WC if $M(a_i, a_j) \leq u(a_i, a_j) < D(a_i, a_j)$, WI if $d(a_i, a_j) \leq u(a_i, a_j) < M(a_i, a_j)$, and SI if $u(a_i, a_j) < d(a_i, a_j)$.

Definition 2. At run-time execution stage, at checkpoint a_p between a_i and a_j , $U(a_i, a_j)$ is said to be of SC if $R(a_i, a_p) + D(a_{p+1}, a_j) \leq u(a_i, a_j)$, WC if $R(a_i, a_p) + M(a_{p+1}, a_j) \leq u(a_i, a_j) < R(a_i, a_p) + D(a_{p+1}, a_j)$, WI if $R(a_i, a_p) + d(a_{p+1}, a_j) \leq u(a_i, a_j) < R(a_i, a_p) + M(a_{p+1}, a_j)$, and SI if $u(a_i, a_j) < R(a_i, a_p) + d(a_{p+1}, a_j)$.

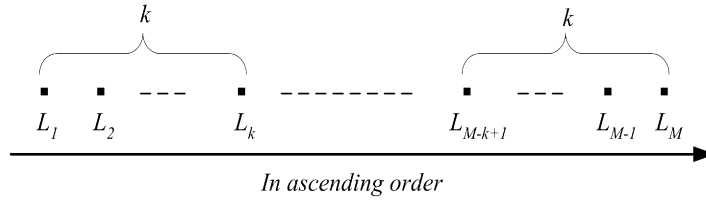


Fig. 1. Relationship between L_k and L_{M-k+1} .

In Definition 2, a checkpoint is an activity point where temporal verification must be conducted [6–8].

According to [5], along scientific workflow execution, for SC, we do not need to do anything as the corresponding upper bound constraints can be kept. For WC, by utilising the possible time redundancy of succeeding activity execution, i.e. the time saved by the execution of each succeeding activity from its pre-set maximum duration, the corresponding upper bound constraints may still be kept. Specific methods for utilising the possible time redundancy can be found in [5]. For WI and SI, basically for most cases, the corresponding upper bound constraints cannot be kept. Consequently, the corresponding exception handling needs to be triggered to adjust them to SC or WC. Specific exception handling methods can be borrowed and adapted from [15,26].

Since WI and SI are adjusted to SC or WC by their respective exception handling, localising fine-grained upper bound constraints actually focuses SC and WC. In this paper, we focus on SC only. The corresponding discussion for WC is similar, hence omitted.

3. Localising fine-grained upper bound constraints

3.1. Assigning fine-grained upper bound constraints at build-time stage

In Section 3.1.1, we detail the assigning process. Then, in Section 3.1.2, we present the overall assigning algorithm.

3.1.1. Assigning process

We denote a concerned coarse-grained upper bound constraint as U and its value as $u(U)$. We suppose that U cover T activities. For simplicity, we number these activities from a_1 , i.e. they are a_1, a_2, \dots, a_T . Since we will only focus on SC, we can suppose that U be of SC.

Within U , based on the past execution history, we can summarise those scientific workflow path slots where temporal violations often happen. Accordingly, at each such slot we should set a fine-grained upper bound constraint. We suppose there be N such scientific workflow path slots. Correspondingly, we need to set N fine-grained upper bound constraints. We denote them as U_1, U_2, \dots, U_N , and their values as $u(U_1), u(U_2), \dots, u(U_N)$. We suppose U_i cover M_i activities, denoted as a_{ij} ($j = 1, 2, 3, \dots, M_i$). Among all of U_i ($i = 1, 2, 3, \dots, N$), there may be some fine-grained upper bound constraints which cover some activities in common. Since U is of SC, according to Definition 1, we have a time redundancy: $u(U) - D(a_1, a_T)$. This time redundancy can be used to tolerate certain time deviation incurred by abnormal scientific workflow execution. Based on the time redundancy, we can derive U_1, U_2, \dots, U_N as follows.

Suppose there be M activities in total covered by U_1, U_2, \dots, U_N . Note that M may not be equal to $M_1 + M_2 + M_3 + \dots + M_N$ because some of U_1, U_2, \dots, U_N may have some activities in common. Among M activities, we first sort all $D(a_s) - M(a_s)$ ($s = 1, 2, 3, \dots, M$) in ascending order to get a sorting list. We denote the list as L and the items in L as L_1, L_2, \dots, L_M . We also denote the numbers of activities corresponding to L_1, L_2, \dots, L_M as l_1, l_2, \dots, l_M . If $D(a_s) - M(a_s)$ is ranked No. k in L , i.e. L_k , then we propose formula (1) below to allocate $u(U) - D(a_1, a_T)$ to each of M activities. We denote the time quota allocated to a_s as $TQ(a_s)$.

$$TQ(a_s) = [u(U) - D(a_1, a_T)] \frac{L_{M-k+1}}{\sum_{l=1}^M [D(a_l) - M(a_l)]} \quad (1 \leq k \leq M) \quad (1)$$

The relationship between L_k and L_{M-k+1} is depicted in Fig. 1.

We now further explain formula (1). In formula (1), we allocate $u(U) - D(a_1, a_T)$ to activities covered by U_1, U_2, \dots, U_N based on the difference between activity maximum duration and activity mean duration. The activity with a bigger difference will be allocated a smaller quota of $u(U) - D(a_1, a_T)$. This is because statistically, an activity could be completed around its mean duration. Therefore, the activity with a bigger difference between its maximum duration and its mean duration has more redundant time to compensate the possible time deviation incurred by abnormal scientific workflow execution. Hence, we should allocate a smaller quota to it.

After we allocate $u(U) - D(a_1, a_T)$ to activities covered by U_1, U_2, \dots, U_N , each activity a_{ij} ($i = 1, 2, 3, \dots, N; j = 1, 2, 3, \dots, M_i$) will carry a time quota. We then can derive the values of U_1, U_2, \dots, U_N . Considering U_i , we derive its value by formula (2) below.

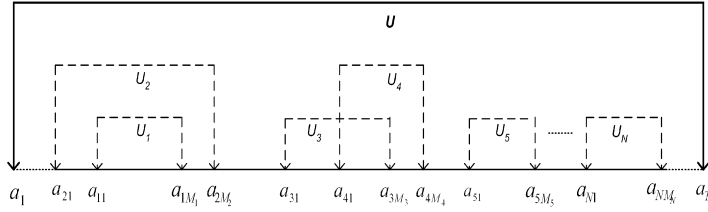


Fig. 2. A sample relationship between U and U_1, U_2, \dots , and U_N .

$$u(U_i) = \sum_{j=1}^{M_i} [TQ(a_{ij}) + D(a_{ij})] \quad (i = 1, 2, 3, \dots, N) \tag{2}$$

A sample relationship between U and U_1, U_2, \dots , and U_N is depicted in Fig. 2.

We now further explain formula (2). Considering a_{ij} covered by U_i , $TQ(a_{ij})$ is the time quota allocated to it and $D(a_{ij})$ is its maximum duration. Apparently, as long as a_{ij} can be completed within $TQ(a_{ij}) + D(a_{ij})$, the execution of a_{ij} is normal and would not impact temporal correctness of overall scientific workflow execution. Correspondingly, for all M_i activities covered by U_i together, i.e. a_{ij} ($j = 1, 2, 3, \dots, M_i$), we can see: if all M_i activities can be completed within $\sum_{j=1}^{M_i} [TQ(a_{ij}) + D(a_{ij})]$, then their execution would be normal and would not impact temporal correctness of overall scientific workflow execution. Therefore, we set the value of $u(U_i)$ to $\sum_{j=1}^{M_i} [TQ(a_{ij}) + D(a_{ij})]$, i.e. as shown in formula (2).

To demonstrate the applicability of formulas (1) and (2), we must prove that all assigned fine-grained upper bound constraints are also of SC. Otherwise, the assigning process may cause some new temporal violations and hence should not be deployed. We derive Theorem 1 to support the applicability.

Theorem 1. Let U be of SC. If we allocate its time redundancy $u(U) - D(a_1, a_T)$ according to formula (1) and assign fine-grained upper bound constraints according to formula (2), then, if U is of SC, all fine-grained upper bound constraints are also of SC.

Proof. Considering a fine-grained upper bound constraint, say U_i . If U is of SC, $u(U) \geq D(a_1, a_T)$, i.e. $u(U) - D(a_1, a_T) \geq 0$. Then, according to formula (1), $TQ(a_{ij})$ is a share of $u(U) - D(a_1, a_T)$. Hence, $TQ(a_{ij}) \geq 0$ ($j = 1, 2, 3, \dots, M_i$). With formula (2), we have: $u(U_i) = \sum_{j=1}^{M_i} [TQ(a_{ij}) + D(a_{ij})] \geq \sum_{j=1}^{M_i} D(a_{ij}) = D(a_{i1}, a_{iM_i})$, i.e. we have inequation (3) below.

$$u(U_i) \geq D(a_{i1}, a_{iM_i}) \tag{3}$$

According to Definition 1, inequation (3) means that U_i is of SC.

Thus, in overall terms, the theorem holds. □

3.1.2. Assigning algorithm

Based on the discussion of Section 3.1.1, we can derive an algorithm for assigning fine-grained upper bound constraints at build-time stage. The main part of the algorithm is depicted in Algorithm 1.

3.2. Adjusting fine-grained upper bound constraints at run-time execution stage

At run-time execution stage, activity completion duration is uncertain and dynamically changing with the current system load. As a result, there may be a time saving or deficit by each activity execution. With the time saving, the remaining fine-grained upper bound constraints would have more time to tolerate possible time deviation of scientific workflow execution. In contrast, with the time deficit, there would be less time for the remaining fine-grained upper bound constraints to tolerate possible time deviation. That is to say, the time saving or deficit could be used to adjust the values of fine-grained upper bound constraints. Meanwhile, since the fine-grained upper bound constraints are temporarily assigned rather than pre-set by users based on user needs, they should be adjusted based on the time saving or deficit on the fly.

Considering an activity, say a_p , there are two cases after its execution. One is that its completion duration is less than or equal to its maximum duration, i.e. $R(a_p) \leq D(a_p)$. In this case, there is a time saving: $D(a_p) - R(a_p)$. The other is $R(a_p) > D(a_p)$ where there is a time deficit: $R(a_p) - D(a_p)$. For the first case, we can enlarge the values of remaining fine-grained upper bound constraints. For the second case, we have to reduce their values. Meanwhile, we should pay attention to the user-set coarse-grained upper bound constraint U . If U is still of SC after the execution of a_p , we simply adjust remaining fine-grained upper bound constraints without the consideration of U . Otherwise, new U might be introduced by corresponding exception handling. With new U , the adjustment process would be to re-set new values to the remaining fine-grained upper bound constraints based on the new time redundancy of U . This would be similar to the assigning process of build-time stage.

In the following, we discuss the two cases in Sections 3.2.1 and 3.2.2, respectively. In Section 3.2.3, we present the overall algorithm for dynamically adjusting fine-grained upper bound constraints at run-time execution stage. When scientific workflow execution arrives at a_p , some of fine-grained upper bound constraints have been completed. We suppose

Input	ArrayUA: an array of all T activities covered by U . ArrayMA: an array of all M activities covered by all time slots where fine-grained upper bound constraints need to be assigned. Maximum and mean durations of all activities in ArrayUA and ArrayMA. U : a coarse-grained upper bound constraint.
Output	Fine-grained upper bound constraints within U .
Step 1	Sorting all $D(a_s) - M(a_s)$ of M activities in ArrayMA.
	1.1. For each activity from ArrayMA, say a_s , compute $D(a_s) - M(a_s)$. 1.2. Sort all $D(a_s) - M(a_s)$ in ascending order to ArrayLA. Suppose $D(a_s) - M(a_s)$ be ranked No. k in ArrayLA.
Step 2	Allocating the time redundancy of upper bound constraint U , i.e. $u(U) - D(a_1, a_T)$, to all M activities in ArrayMA.
	2.1. Based on ArrayUA, compute $u(U) - D(a_1, a_T)$. 2.2. For each of M activities in ArrayMA, say a_s ($s = 1, 2, 3, \dots, M$), compute its time quota $TQ(a_s)$ as follows, i.e. formula (1).
	$TQ(a_s) = [u(U) - D(a_1, a_T)] \frac{L_{M-k+1}}{\sum_{l=1}^M [D(a_l) - M(a_l)]} \quad (1 \leq k \leq M)$
Step 3	Computing values of fine-grained upper bound constraints.
	3.1. For each of fine-grained upper bound constraints, say U_i at the i th time slot, we compute its value as follows, i.e. formula (2).
	$u(U_i) = \sum_{j=1}^{M_i} [TQ(a_{ij}) + D(a_{ij})] \quad (i = 1, 2, 3, \dots, N)$

Algorithm 1. Assigning fine-grained upper bound constraints at build-time stage.

that the remaining fine-grained upper bound constraints are U_k, U_{k+1}, \dots , and U_N . The unexecuted activities covered by U_k, U_{k+1}, \dots , and U_N form a subset of L . We denote the subset as L -sub and the number of activities in L -sub as R . In addition, among U_k, U_{k+1}, \dots , and U_N , we suppose those covering a_p be U_k, U_{k+1}, \dots , and U_{N_p} ($N_p \leq N$). We also suppose a_p be the No. j_p activity among all activities covered by U_j , i.e. a_{j_p} ($k \leq j \leq N_p$).

3.2.1. Handling time saving case

This is the case of $R(a_p) \leq D(a_p)$. After the execution of a_p , we have a time saving: $D(a_p) - R(a_p)$. We derive Theorem 2 first. Based on Theorem 2, we are able to use the time saving to adjust the remaining fine-grained upper bound constraints U_k, U_{k+1}, \dots , and U_N without affecting U .

Theorem 2. Let U be of SC before the execution of a_p . Then, if $R(a_p) \leq D(a_p)$, U is still of SC after the execution of a_p .

Proof. Suppose U be of SC before the execution of a_p . Then, according to Definition 2, we have: $R(a_1, a_{p-1}) + D(a_p, a_T) \leq u(U)$. Meanwhile, we have: $R(a_1, a_p) + D(a_{p+1}, a_T) = R(a_1, a_{p-1}) + R(a_p) + D(a_{p+1}, a_T)$. If $R(a_p) \leq D(a_p)$, then, $R(a_1, a_{p-1}) + R(a_p) + D(a_{p+1}, a_T) \leq R(a_1, a_{p-1}) + D(a_p) + D(a_{p+1}, a_T) = R(a_1, a_{p-1}) + D(a_p, a_T) \leq u(U)$, i.e. we have in equation (4) below.

$$R(a_1, a_p) + D(a_{p+1}, a_T) \leq u(U) \quad (4)$$

According to Definition 2, inequation (4) means that U is still of SC after the execution of a_p . Thus, in overall terms, the theorem holds. \square

According to Theorem 2, if $R(a_p) \leq D(a_p)$, we can use $D(a_p) - R(a_p)$ to enlarge the values of the remaining fine-grained upper bound constraints directly without considering the situation where new U is introduced.

We now investigate how to allocate $D(a_p) - R(a_p)$ to U_k, U_{k+1}, \dots , and U_N so that each of them has more time redundancy to tolerate possible time deviation of scientific workflow execution. To do so, we allocate $D(a_p) - R(a_p)$ to unexecuted activities covered by U_k, U_{k+1}, \dots , and U_N , i.e. the R activities in L -sub. We denote their numbers are s_1, s_2, \dots , and s_R . Considering an activity among the R ones, say a_r , if $D(a_r) - M(a_r)$ is ranked No. n in L -sub, i.e. L_n , we propose formula (5) below to allocate $D(a_p) - R(a_p)$ to a_r . We denote the extra quota allocated to a_r as $EQ(a_r)$.

$$EQ(a_r) = [D(a_p) - R(a_p)] \frac{L_{R-n+1}}{\sum_{l=s_1}^{s_R} [D(a_l) - M(a_l)]} \quad (1 \leq r \leq R) \quad (5)$$

The relationship between L_n and L_{R-n+1} is similar to that between L_k and L_{M-k+1} as shown in Fig. 1.

The detailed explanation for formula (5) is similar to that for formula (1), hence omitted here.

After we allocate $D(a_p) - R(a_p)$ to the unexecuted activities covered by U_k, U_{k+1}, \dots , and U_N , each of them, say a_{rs} ($r = k, k + 1, \dots, N_p$ and $s = j_p j_p + 1, \dots, M_r$; $r = N_p + 1, \dots, N$ and $s = 1, 2, \dots, M_r$) will carry an extra quota. Based on this, we can derive new values of U_k, U_{k+1}, \dots , and U_N . Considering U_j ($j = k, k + 1, \dots, N$), we propose formula (6) to compute its new value if $k \leq j \leq N_p$. And we propose formula (7) to compute its new value if $N_p < j \leq N$.

$$u(U_j) = u(U_j) + \sum_{s=j_p+1}^{M_j} EQ(a_{js}) \quad (j = k, k + 1, \dots, N_p) \tag{6}$$

$$u(U_j) = u(U_j) + \sum_{s=1}^{M_j} EQ(a_{js}) \quad (j = N_p + 1, \dots, N) \tag{7}$$

We now further explain formulas (6) and (7). After we allocate $D(a_p) - R(a_p)$ to the unexecuted activities covered by U_k, U_{k+1}, \dots , and U_N , U_j ($j = k, k + 1, \dots, N$) will receive an overall time quota which is the sum of all extra quotas carried by all unexecuted activities covered by U_j . Accordingly, its new value is equal to its old value plus the overall time quota. There are two cases for computing the overall time quota. One is that U_j covers a_p , i.e. $j = k, k + 1, \dots, N_p$. The other is that U_j does not cover a_p , i.e. $j = N_p + 1, \dots, N$.

Case 1. The unexecuted activities of U_j are a_{js} where $s = j_p + 1, j_p + 2, \dots, M_j$. Correspondingly, the overall time quota is $\sum_{s=j_p+1}^{M_j} EQ(a_{js})$. Therefore, we propose formula (6) to compute the new value of U_j .

Case 2. The unexecuted activities of U_j are a_{js} where $s = 1, 2, \dots, M_j$. Correspondingly, the overall time quota is $\sum_{s=1}^{M_j} EQ(a_{js})$. Therefore, we propose formula (7) to compute the new value of U_j .

3.2.2. Handling time deficit case

This is the case of $R(a_p) > D(a_p)$. The execution of a_p causes a time deficit: $R(a_p) - D(a_p)$. Correspondingly, we need to adjust the remaining fine-grained upper bound constraints, i.e. U_k, U_{k+1}, \dots , and U_N . We first derive Theorem 3. Based on Theorem 3, there are two situations for adjusting U_k, U_{k+1}, \dots , and U_N under the condition where $R(a_p) > D(a_p)$.

Theorem 3. Let U be of SC before the execution of a_p . Then, if $R(a_p) > D(a_p)$, U may or may not be of SC after the execution of a_p .

Proof. Suppose U is of SC before the execution of a_p . Then, according to Definition 2, we have inequation (8) below.

$$R(a_1, a_{p-1}) + D(a_p, a_T) \leq u(U) \tag{8}$$

In addition, we have: $R(a_1, a_p) + D(a_{p+1}, a_T) = R(a_1, a_{p-1}) + R(a_p) + D(a_{p+1}, a_T)$. If $R(a_p) > D(a_p)$, then, $R(a_1, a_{p-1}) + R(a_p) + D(a_{p+1}, a_T) > R(a_1, a_{p-1}) + D(a_p) + D(a_{p+1}, a_T) = R(a_1, a_{p-1}) + D(a_p, a_T)$. Hence, we have inequation (9) below.

$$R(a_1, a_{p-1}) + D(a_p, a_T) < R(a_1, a_p) + D(a_{p+1}, a_T) \tag{9}$$

According to Definition 2, for U to be of SC after the execution of a_p , we must ensure that inequation (10) below should hold.

$$R(a_1, a_p) + D(a_{p+1}, a_T) \leq u(U) \tag{10}$$

However, based on inequations (8) and (9) which we only have, we cannot ensure inequation (10). In fact, depending on how much $R(a_p)$ is greater than $D(a_p)$, inequation (10) may or may not hold. That is to say, U may or may not be of SC after the execution of a_p .

Thus, in overall terms, the theorem holds. \square

According to Theorem 3, one situation is that U is still of SC after the execution of a_p , which will be discussed in Section 3.2.2.1. The other situation is that U is not of SC after the execution of a_p , which will be discussed in Section 3.2.2.2.

3.2.2.1. Being of SC after execution of a_p Under the condition of $R(a_p) > D(a_p)$, this situation means that the time deficit $R(a_p) - D(a_p)$ incurred by the execution of a_p does not impact the overall temporal correctness of scientific workflow execution. In other words, the time deficit could be counteracted locally within U . Hence, we can adjust the remaining fine-grained upper bound constraints to tolerate the time deficit without affecting U .

Similar to Section 3.2.1, we allocate the time deficit $R(a_p) - D(a_p)$ to unexecuted activities covered by U_k, U_{k+1}, \dots , and U_N , i.e. the R activities in L -sub. For a_r , we denote the deficit quota allocated to it as $DQ(a_r)$. Then, similar to formula (5), we propose formula (11) below to derive $DQ(a_r)$.

$$DQ(a_r) = [R(a_p) - D(a_p)] \frac{L_{R-n+1}}{\sum_{l=s_1}^{s_R} [D(a_l) - M(a_l)]} \quad (1 \leq r \leq R) \quad (11)$$

The detailed explanation for formula (11) is similar to that for formula (1), hence omitted here.

After we allocate $R(a_p) - D(a_p)$ to the unexecuted activities covered by U_k, U_{k+1}, \dots , and U_N , each of them, say a_{rs} ($r = k, k+1, \dots, N_p$ and $s = j_p j_p + 1, \dots, M_r$; $r = N_p + 1, \dots, N$ and $s = 1, 2, \dots, M_r$) will hold a deficit quota. The new value of U_j ($j = k, k+1, \dots, N$), would be equal to its old value minus total deficit quotas allocated to all unexecuted activities covered by U_j . Similar to formulas (6) and (7), we propose formulas (12) and (13) below to derive the new value of U_j .

$$u(U_j) = u(U_j) - \sum_{s=j_p+1}^{M_j} DQ(a_{js}) \quad (j = k, k+1, \dots, N_p) \quad (12)$$

$$u(U_j) = u(U_j) - \sum_{s=1}^{M_j} DQ(a_{js}) \quad (j = N_p + 1, \dots, N) \quad (13)$$

The detailed explanations for formulas (12) and (13) are similar to those for formulas (6) and (7) respectively, hence omitted here.

3.2.2.2. Not being of SC after execution of a_p Under the condition of $R(a_p) > D(a_p)$, this situation means that the time deficit $R(a_p) - D(a_p)$ incurred by the execution of a_p will impact the overall temporal correctness of scientific workflow execution. Then, corresponding exception handling would be triggered. We assume that the handling result is to introduce new U with some price as this is very common in reality [15,26]. To distinguish with U , we denote new U as U' and its value as $u(U')$.

With U' , the previous values of the remaining fine-grained upper bound constraints U_k, U_{k+1}, \dots , and U_N would not be useable. Therefore, we must remove all types of quotas which have been allocated to the unexecuted activities covered by U_k, U_{k+1}, \dots , and U_N . Then, we set new quotes to the unexecuted activities based on U' to obtain new values of U_k, U_{k+1}, \dots , and U_N .

With U' , we have a new time redundancy: $u(U') - [R(a_1, a_p) + D(a_{p+1}, a_T)]$. This new redundancy could be used to tolerate the future time deviation which might be incurred by the succeeding scientific workflow execution. We allocate this new redundancy to unexecuted activities covered by U_k, U_{k+1}, \dots , and U_N , i.e. the R activities in L -sub. For a_r where $D(a_r) - M(a_r)$ is ranked No. n in L -sub, i.e. L_n , we denote the new quota allocated to it as $NQ(a_r)$. Then, similar to formula (1), we propose formula (14) below to derive $NQ(a_r)$.

$$NQ(a_r) = \{u(U') - [R(a_1, a_p) + D(a_{p+1}, a_T)]\} \frac{L_{R-n+1}}{\sum_{l=s_1}^{s_R} [D(a_l) - M(a_l)]} \quad (1 \leq r \leq R) \quad (14)$$

The detailed explanation for formula (14) is similar to that for formula (1), hence omitted here.

After we allocate the new time redundancy to unexecuted activities covered by U_k, U_{k+1}, \dots , and U_N , each of them will carry a new time quota. The new value of U_j ($j = k, k+1, \dots, N$), would be equal to the sum of all new quotas and all maximum durations of all unexecuted activities of U_j . Based on formulas (2), (6) and (7), we propose formulas (15) and (16) below to derive the new value of U_j .

$$u(U_j) = \sum_{s=j_p+1}^{M_j} [NQ(a_{js}) + D(a_{js})] \quad (j = k, k+1, \dots, N_p) \quad (15)$$

$$u(U_j) = \sum_{s=1}^{M_j} [NQ(a_{js}) + D(a_{js})] \quad (j = N_p + 1, \dots, N) \quad (16)$$

The detailed explanation for formula (15) or (16) is similar to that for formula (2). The detailed explanation for why we use two cases, i.e. ($j = k, k+1, \dots, N_p$) and ($j = N_p + 1, \dots, N$), is similar to that for formulas (6) and (7).

If we have a close look at formula (15), we can see that we do not consider those activities which are completed but covered by U_k, U_{k+1}, \dots , and U_{N_p} ($N_p \leq N$). This is because fine-grained upper bound constraints are temporarily set rather than by users. When we set them again based on U' , the process is that we remove all previously assigned or allocated quotas and set new ones. Therefore, previously completed activities need not be taken into consideration. Correspondingly, the new start point of U_k, U_{k+1}, \dots , and U_{N_p} is just a_p .

3.2.3. Adjusting algorithm

Based on the discussion in Sections 3.2.1 and 3.2.2, we can derive an overall algorithm for dynamically adjusting fine-grained upper bound constraints at run-time execution stage. The main part of the algorithm is depicted in Algorithm 2.

Input	<p>U: a coarse-grained upper bound constraint. a_p: current position where scientific workflow execution arrives at. All remaining fine-grained upper bound constraints U_k, U_{k+1}, \dots, and U_N after a_p. ArrayUA: an array of all T activities covered by U. ArrayRA: an array of all remaining unexecuted activities covered by U_k, U_{k+1}, \dots, and U_N. Maximum, minimum and mean durations of all activities.</p>
Output	New fine-grained upper bound constraints if adjustment happens.
Step 1	<p>If $R(a_p) \leq D(a_p)$: adjusting remaining fine-grained upper bound constraints without affecting U at checkpoint a_p.</p> <ol style="list-style-type: none"> For each activity from ArrayRA, say a_r, compute $D(a_r) - M(a_r)$. Sort all $D(a_r) - M(a_r)$ in ascending order to ArraySA. Suppose $D(a_r) - M(a_r)$ be ranked No. n in ArraySA. Allocate $D(a_p) - R(a_p)$ to each of the R activities in ArrayRA, say a_r ($r = 1, 2, 3, \dots, R$); compute its extra quota $EQ(a_r)$ as follows, i.e. formula (5). $EQ(a_r) = [D(a_p) - R(a_p)] \frac{L_{R-n+1}}{\sum_{l=s_1}^{s_R} [D(a_l) - M(a_l)]} \quad (1 \leq r \leq R)$ For each of U_k, U_{k+1}, \dots, and U_N, say U_j ($j = k, k + 1, \dots, N$), compute its new value as follows, i.e. formulas (6) and (7). $u(U_j) = u(U_j) + \sum_{s=j_p+1}^{M_j} EQ(a_{js}) \quad (j = k, k + 1, \dots, N_p)$ $u(U_j) = u(U_j) + \sum_{s=1}^{M_j} EQ(a_{js}) \quad (j = N_p + 1, \dots, N)$
Step 2	<p>If $R(a_p) > D(a_p)$ and U is of SC: adjusting remaining fine-grained upper bound constraints without affecting U.</p> <ol style="list-style-type: none"> For each activity from ArrayRA, say a_r, compute $D(a_r) - M(a_r)$. Sort all $D(a_r) - M(a_r)$ in ascending order to ArraySA. Suppose $D(a_r) - M(a_r)$ be ranked No. n in ArraySA. Allocate $R(a_p) - D(a_p)$ to each of the R activities in ArrayRA, say a_r ($r = 1, 2, 3, \dots, R$); compute its deficit quota $DQ(a_r)$ as follows, i.e. formula (11). $DQ(a_r) = [R(a_p) - D(a_p)] \frac{L_{R-n+1}}{\sum_{l=s_1}^{s_R} [D(a_l) - M(a_l)]} \quad (1 \leq r \leq R)$ For each of U_k, U_{k+1}, \dots, and U_N, say U_j ($j = k, k + 1, \dots, N$), compute its new value as follows, i.e. formulas (12) and (13). $u(U_j) = u(U_j) - \sum_{s=j_p+1}^{M_j} DQ(a_{js}) \quad (j = k, k + 1, \dots, N_p)$ $u(U_j) = u(U_j) - \sum_{s=1}^{M_j} DQ(a_{js}) \quad (j = N_p + 1, \dots, N)$
Step 3	<p>If $R(a_p) > D(a_p)$ and U is not of SC: adjusting remaining fine-grained upper bound constraints with new U, i.e. U'.</p> <ol style="list-style-type: none"> Remove any types of quotas which have been allocated to the unexecuted activities covered by U_k, U_{k+1}, \dots, and U_N. For each activity from ArrayRA, say a_r, compute $D(a_r) - M(a_r)$. Sort all $D(a_r) - M(a_r)$ in ascending order to ArraySA. Suppose $D(a_r) - M(a_r)$ be ranked No. n in ArraySA. Based on ArrayUA, Compute $u(U') - [R(a_1, a_p) + D(a_{p+1}, a_T)]$.

- 3.5. For each of the R activities in ArrayRA, say a_r ($r = 1, 2, 3, \dots, R$); compute its new time quota $NQ(a_r)$ according to formula (14).

$$NQ(a_r) = \{u(U') - [R(a_1, a_p) + D(a_{p+1}, a_T)]\} \frac{L_{R-n+1}}{\sum_{l=s_1}^{s_R} [D(a_l) - M(a_l)]} \quad (1 \leq r \leq R)$$

- 3.6. For each of U_k, U_{k+1}, \dots , and U_N , say U_j ($j = k, k+1, \dots, N$), compute its new value as follows, i.e. formulas (15) and (16).

$$u(U_j) = \sum_{s=j_p+1}^{M_j} [NQ(a_{js}) + D(a_{js})] \quad (j = k, k+1, \dots, N_p)$$

$$u(U_j) = \sum_{s=1}^{M_j} [NQ(a_{js}) + D(a_{js})] \quad (j = N_p + 1, \dots, N)$$

- 3.7. Change the start activity of U_k, U_{k+1}, \dots , and U_N to a_p .

Algorithm 2. Adjusting fine-grained upper bound constraints at run-time execution stage.

4. Comparison and quantitative evaluation

With fine-grained upper bound constraints, we can control scientific workflow execution locally at various activities. When a temporal violation happens, we can try to handle it locally rather than globally within U . Local handling would affect fewer activities and hence is more cost effective than global handling. Now, we conduct a quantitative analysis so that we can have a clear picture on how the introduction of fine-grained upper bound constraints can achieve better handling cost effectiveness.

Within U , we suppose there be S fine-grained upper bound constraints. For a fine-grained upper bound constraint, we suppose that statistically there be X temporal violations which can be handled within it. For simplicity, we assume that the number of activities between any two adjacent fine-grained upper bound constraints be the same, denoted as Q , and each fine-grained upper bound constraint cover the same number of activities, denoted as P . In addition, we also assume that X temporal violations happen respectively at the first X activities of each fine-grained upper bound constraint. We denote the exception handling cost for an activity as C . We denote the exception handling cost based on U as C_{global} , and that based on fine-grained upper bound constraints as C_{local} . Then, the improvement on overall cost effectiveness is reflected by how C_{local} is less than C_{global} .

For the k th temporal violation in the i th fine-grained upper bound constraint, the exception handling cost based on U is $[i * Q + (i - 1) * P + k] * C$ while the exception handling cost based on fine-grained upper bound constraints is $k * C$. Therefore, for S fine-grained upper bound constraints in total, we have formulas (17) and (18) below.

$$C_{local} = S * \sum_{k=1}^X k * C \quad (17)$$

$$C_{global} = \sum_{i=1}^S \sum_{k=1}^X \{[i * Q + (i - 1) * P + k] * C\} \quad (18)$$

We now take a set of specific values to see how formulas (17) and (18) perform. We suppose that $P = 3$, $Q = 2$, $X = 2$, C be equal to 1 cost unit. We also suppose that S can change from 0 to 20. The selection of these specific values is rather random and does not affect our analysis because what we want to see is the trend of how C_{local} and C_{global} change based on S . With S changing, we list corresponding C_{local} and C_{global} in Fig. 3.

According to Fig. 3, we can see that with S increasing, both C_{local} and C_{global} are increasing. However, their increase rates are quite different. C_{local} increases slowly while C_{global} increases dramatically. Particularly, when S is getting larger, C_{global} is getting much greater than C_{local} . In real-world scientific workflow systems, scientific workflows are normally very complicated and contain hundreds of thousands of activities [11,20,24]. Consequently, a scientific workflow execution normally lasts a long time [11,20,24]. Therefore, to better control local scientific workflow execution, a good number of fine-grained upper bound constraints are often needed. That is to say, in the real-world scientific workflow systems, normally, S is a large number. Therefore, in overall terms, we can conclude that introducing a series of fine-grained upper bound constraints can achieve much better cost effectiveness.

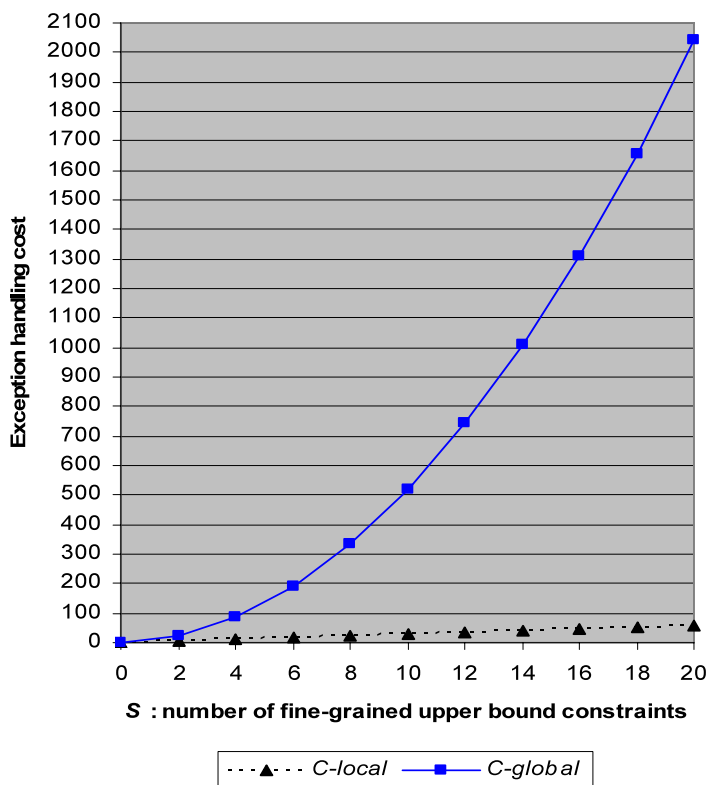


Fig. 3. Change trend of C_{local} and C_{global} by number of fine-grained upper bound constraints – S .

5. Conclusions and future work

Having a few coarse-grained upper bound constraints in a scientific workflow is simple and easy from the perspective of users. However, it is not sufficient to control and monitor scientific workflow execution locally at various activities. Consequently, we are not able to detect temporal violations in time and handle them locally for better handling cost effectiveness.

In this paper, we have investigated how to localise fine-grained upper bound constraints within user-set coarse-grained ones so that we can obtain a series of upper bound constraints. The corresponding build-time assigning algorithm and run-time adjusting algorithm have been developed accordingly. A quantitative evaluation has been conducted to demonstrate that with fine-grained upper bound constraints we can improve cost effectiveness of handling temporal violations significantly than with coarse-grained upper bound constraints. As stated in Section 1, such results can be equally applied to all types of temporal constraints.

With these contributions, we can further investigate temporal exception handling approaches when a temporal constraint is violated.

Acknowledgments

The work reported in this paper is partly supported by Australian Research Council Linkage Project under grant No. LP0990393.

References

- [1] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, A.P. Barros, Workflow patterns, *Distrib. Parallel Databases* 14 (1) (2003) 5–51.
- [2] A. Abramson, J. Kommineni, J.L. McGregor, J. Katzfey, An atmospheric sciences workflow and its implementation with Web services, *Future Generation Comput. Syst.* 21 (1) (2005) 69–78.
- [3] J. Alhiyafi, C. Sabesan, S. Lu, J.L. Ram, RECOMBFLOW: A scientific workflow environment for Intragenomic Gene Conversion analysis in bacterial genomes, including the pathogen *Streptococcus pyogenes*, *Int. J. Bioinform. Res. Appl.* 5 (1) (2009) 1–19.
- [4] I. Brandic, S. Pillana, S. Benkner, Specification, planning, and execution of QoS-aware Grid workflows within the Amadeus environment, *Concurrency Computation: Practice & Experience* 20 (4) (2008) 331–345.
- [5] J. Chen, Y. Yang, Multiple states based temporal consistency for dynamic verification of fixed-time constraints in Grid workflow systems, *Concurrency Computation: Practice & Experience* 19 (7) (2007) 965–982.
- [6] J. Chen, Y. Yang, Activity completion duration based checkpoint selection for dynamic verification of temporal constraints in Grid workflow systems, *Int. J. High Performance Comput. Appl.* 22 (3) (2008) 319–329.

- [7] J. Chen, Y. Yang, Adaptive selection of necessary and sufficient checkpoints for dynamic verification of temporal constraints in Grid workflow systems, *ACM Trans. Autonomous Adaptive Syst.* 2 (2) (2007), Article 6.
- [8] J. Chen, Y. Yang, Temporal dependency based checkpoint selection for dynamic verification of temporal constraints in scientific workflow systems, *ACM Trans. Software Engrg. Methodol.*, in press (accepted on 17 June 2009), <http://www.swinflow.org/papers/TOSEM.pdf>, accessed on 20 July 2009.
- [9] S. Chinn, G. Madey, Temporal representation and reasoning for workflow in engineering design change review, *IEEE Trans. Eng. Manage.* 47 (4) (2000) 485–492.
- [10] D. Cybok, A Grid workflow infrastructure, *Concurrency Comput. Pract. Ex.* 18 (10) (2006) 1243–1254, special issue on workflow in Grid systems.
- [11] K. Daisuke, C. Runyue, C.H. Luis, Gravitational stability of circumnuclear disks in elliptical galaxies, *Astrophysical J.* 669 (1) (2007) 232–240.
- [12] E. Deelman, A.L. Chervenak, Data management challenges of data-intensive scientific workflows, in: *Proceedings of the 8th IEEE International Symposium on Cluster Computing and the Grid*, May 2008, Lyon, France, IEEE CS Press, pp. 687–692.
- [13] J. Eder, E. Panagos, M. Rabinovich, Time constraints in workflow systems, in: *Proceedings of the 11th International Conference on Advanced Information Systems Engineering*, Heidelberg, Germany, June 1999, in: *Lecture Notes in Comput. Sci.*, vol. 1626, Springer-Verlag, 1999, pp. 286–300.
- [14] Y. Gil, E. Deelman, M. Ellisman, T. Fahringer, G. Fox, D. Gannon, C. Goble, M. Livny, L. Moreau, J. Myers, Examining the challenges of scientific workflows, *IEEE Comput.* 40 (12) (2007) 24–32.
- [15] C. Hagen, G. Alonso, Exception handling in workflow management systems, *IEEE Trans. Software Eng.* 26 (10) (2000) 943–958.
- [16] G. Kandaswamy, A. Mandal, D.A. Reed, Fault tolerance and recovery of scientific workflows on computational grids, in: *Proceedings of 8th IEEE International Symposium on Cluster Computing and the Grid*, May 2008, pp. 777–782.
- [17] J. Li, Y. Fan, M. Zhou, Timing constraint workflow nets for workflow analysis, *IEEE Trans. Syst. Man Cybern. Syst. Hum.* 33 (2) (2003) 179–193.
- [18] C. Lin, S. Lu, X. Fei, A. Chebotko, D. Pai, Z. Lai, F. Fotouhi, J. Hua, A reference architecture for scientific workflow management systems and the VIEW SOA solution, *IEEE Trans. Services Comput.* 2 (1) (2009) 79–92.
- [19] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger-Frank, M. Jones, E. Lee, J. Tao, Y. Zhao, Scientific workflow management and the Kepler system, *Concurrency Comput. Pract. Ex.* 18 (10) (2006) 1039–1065.
- [20] P. Maechling, H. Chalupsky, M. Dougherty, E. Deelman, Y. Gil, S. Gullapalli, V. Gupta, C. Kesselman, J. Kim, G. Mehta, B. Mendenhall, T. Russ, G. Singh, M. Spraragen, G. Staples, K. Vahi, Simplifying construction of complex workflows for non-expert users of the Southern California Earthquake Center Community Modeling Environment, *ACM SIGMOD Record* 34 (3) (2005) 24–30.
- [21] N. Mandal, E. Deelman, G. Mehta, M. Su, K. Vahi, Integrating existing scientific workflow systems: The Kepler/Pegasus example, in: *Proceedings of the 16th IEEE International Symposium on High Performance Distributed Computing*, Monterrey, CA, June 2007, pp. 21–28.
- [22] O. Marjanovic, M.E. Orlowska, On modeling and verification of temporal constraints in production workflows, *Knowledge Inform. Syst.* 1 (2) (1999) 157–192.
- [23] T. Oinn, M. Greenwood, M. Addis, M.N. Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, P. Lord, M.R. Pocock, M. Senger, R. Stevens, A. Wipat, C. Wroe, Taverna: Lessons in creating a workflow environment for the life sciences, *Concurrency Comput. Pract. Ex.* 18 (10) (2006) 1067–1100.
- [24] S. Pandey, R. Buyya, Scheduling of scientific workflows on data grids, in: *Proceedings of the 8th IEEE International Symposium on Cluster Computing and the Grid*, May 2008, Lyon, France, IEEE CS Press, pp. 548–553.
- [25] R. Prodan, T. Fahringer, Overhead analysis of scientific workflows in Grid environments, *IEEE Trans. Parallel Distrib. Syst.* 19 (3) (2008) 378–393.
- [26] N. Russell, W.M.P. van der Aalst, A.H.M. ter Hofstede, Workflow exception patterns, in: *Proceedings of the 18th International Conference on Advanced Information Systems Engineering*, in: *Lecture Notes in Comput. Sci.*, vol. 4001, Springer-Verlag, Berlin, 2006, pp. 288–302.
- [27] W. Sadiq, M.E. Orlowska, Analysing process models using graph reduction techniques, *Inform. Syst.* 25 (2) (2000) 117–134.
- [28] J.H. Son, M.H. Kim, Improving the performance of time-constrained workflow processing, *J. Syst. Software* 58 (3) (2001) 211–219.
- [29] SWINDEW-G Team, System architecture of SwinDeW-G, http://www.swinflow.org/docs/System_Architecture.pdf, accessed on 20 July 2009.
- [30] I. Taylor, M. Shields, I. Wang, A. Harrison, The Triana workflow environment: Architecture and applications, in: Ian Taylor, E. Deelman, D. Gannon, M. Shields (Eds.), *Workflows for e-Science*, Springer-Verlag, New York, 2007, pp. 320–339.
- [31] J. Yu, R. Buyya, A taxonomy of scientific workflow systems for Grid computing, *ACM SIGMOD Record* 34 (3) (2005) 44–49.