Future Generation Computer Systems 28 (2012) 1177–1193

Contents lists available at SciVerse ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Modeling and analyzing the impact of authorization on workflow executions☆

Ligang He [a,*], Chenlin Huang [b], Kewei Duan [c], Kenli Li [d], Hao Chen [d], Jianhua Sun [d], Stephen A. Jarvis [a]

[a] Department of Computer Science, University of Warwick, Coventry, UK
[b] Institute of Software, School of Computer Science, National University of Defense Technology, Changsha, China
[c] Department of Computer Science, University of Bath, Bath, UK
[d] School of Computer and Communication, Hunan University, Changsha, China

ARTICLE INFO

Article history:
Received 28 September 2011
Received in revised form
15 February 2012
Accepted 1 March 2012
Available online 5 March 2012

Keywords:
Modeling
Authorization
Workflow
RBAC

ABSTRACT

It has been a subject of a significant amount of research to automate the execution of workflows (or business processes) on computer resources. However, many workflow scenarios still require human involvement, which introduces additional security and authorization concerns. This paper presents a novel mechanism for modeling the execution of workflows with human involvement under Role-based Authorization Control. Our modeling approach applies Colored Timed Petri-Nets to allow various authorization constraints to be modeled, including role, temporal, cardinality, BoD (Binding of Duty), SoD (Separation of Duty), role hierarchy constraints etc. We also model the execution of tasks with different levels of human involvement and as such allow the interactions between workflow authorization and workflow execution to be captured. The modeling mechanism is developed in such a way that the construction of the authorization model for a workflow can be automated. This feature is very helpful for modeling large collections of authorization policies and/or complex workflows. A Petri-net toolkit, the CPN Tools, is utilized in the development of the modeling mechanism and to simulate the constructed models. This paper also presents the methods to analyze and calculate the authorization overhead as well as the performance data in terms of various metrics through the model simulations. Based on the simulation results, this paper further proposes the approaches to improving performance given the deployed authorization policies. This work can be used for investigating the impact of authorization, for capacity planning, for the design of workload management strategies, and also to estimate execution performance, when human resources and authorization policies are employed in tandem.

1. Introduction

Business processes or *workflows* are often used to model enterprise or scientific applications [1–4]. A workflow consists of multiple tasks with ordered execution, i.e., a task can only start execution after another task in the workflow has completed (the former task is called the latter's child). It has received considerable research interest to automate workflow executions on computer resources, which has led in part to BPEL being proposed as a standard for specifying and executing workflows [4]. However, many workflow scenarios still involve human activities and will be comprised of a mixture of human tasks and computing tasks (which we term a *hybrid workflow* in this paper) [5–9]. For example, in IT-based video production workflows [9], human interactions are still required for decision making and artistic choices (e.g., video editing decisions). In mortgage business processes in banks [10], various human tasks (e.g., a manual approval step is required if the mortgage value exceeds some amount) could be involved in order to make the final decisions. Indeed, in many application domains, the completion of a task in a workflow replies on the subjective judgment of human. It would be very difficult, if not possible, to use computers to completely replace humans in such scenarios.

In traditional workflow management systems, human interactions in a workflow are not well supported, and therefore a workflow with human involvement can be regarded as a semi-automated workflow [11]. Motivated by the requirements of integrating human interactions into business processes, research exists to support human tasks in workflow contexts. WS-HumanTask and BPEL4People, which have been proposed to overcome the lack of support for human activities in BPEL [11,10], are the exemplar products of these research efforts. WS-HumanTask and BPEL4People enable the integration of human tasks into business processes, and therefore the executions of the workflows containing human tasks can also be automated [11,10].

Human involvement introduces authorization concerns, requiring restrictions on who is allowed to perform which tasks at what

☆ The preliminary version of this paper was presented in the 8th IEEE Intl. Conf. on Services Computing (SCC'11), 2011.

* Corresponding author. Tel.: +44 24 76573802; fax: +44 24 76573024.
E-mail address: liganghe@dcs.warwick.ac.uk (L. He).

0167-739X/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2012.03.003

time. Research has been conducted to attach authorization information (such as roles and permissions) to activities, and to impose authorization constraints (such as separation of duty) on workflow executions [12–18]. For example, in BPEL4People, authorization concepts such as roles and permissions are defined, and various authorization constraints are supported, including cardinality constraints, separation of duty, binding of duty, etc. The authorization specified in BPEL4People can be categorized as Role-based Authorization Control (RBAC), under which users are assigned to certain roles, while the roles themselves are associated with prescribed permissions.

When we assess resource capacities, or evaluate the performance of workflow executions on supporting platforms, it is often assumed that when a task is allocated to a resource, the resource will accept the task and start the execution once a processor becomes available. However, when human activities and authorization constraints are taken into account, the issue can become complex. The following example illustrates such a situation.

A bank will need both human activities and computing-based activities to support its business. A workflow will typically contain both *Human Tasks* (HT) and *Computing Tasks* (CT): A human task may consist of a person (or a *user* in the RBAC terminology) with an official position (or a *role* in RBAC, e.g., a branch manager) signing a document; a computing task may involve running an application on a computing resource to assess risk for an investment. Further, the computing applications may be hosted in a central resource pool (e.g. a cluster), and the invocation of an application may be automated without human intervention, which we term an *Automated Computing Task* (ACT), or for security reasons, can only be initiated by a user with a certain role and be executed under that role/user, which we term a *Human-aided Computing Task* (HCT). The following authorization constraints are often encountered in such scenarios [17]: (1) Role constraints: A human task may only be performed by a particular role; a computing application may only be invoked by assuming a particular role; (2) Temporal constraints: A role or a user is only activated during certain time intervals (e.g., a staff member only works in morning hours); (3) Cardinality constraints: The maximum number of tasks (computing or other) running simultaneously under a role is N; (4) Separation of Duty constraints: If Task A (HT or CT) is run by a role (or a user), then Task B must not be run by the same role (or user); (5) Binding of Duty constraints: If Task A is run by a role (or user), then Task B must be run by the same role (or user); (6) role hierarchy constraints: if multiple roles are eligible and available to run a task, the task must only assume the role with the least privilege.

In real-world applications, a more complex task may contain both human and computing activities. For example, a task may first require a person to handle the task, and then require a computing application being invoked to compute additional data. Such a complex task can be regarded as the combination of an HT and an ACT/HCT. Therefore, in this paper, we assume that the time spent by a user handling an HCT is negligible.

It is common to find such authorization constraints and interaction between human and automated activities; our domains of interests include healthcare systems [19], the video management domain [9] and the manufacturing community [6,20]. Human intervention and associated authorization clearly affects the processing of tasks and impacts on both application-oriented performance (e.g. mean response time of workflows) and system-oriented performance (e.g. utilization of the computing resource pool). Obtaining these performance data will be critical in capacity planning, designing authorization policies and developing workflow management strategies.

To date, little attention has been paid to investigating performance when running hybrid workflows under deployed authorization policies. The purpose of this paper is to model execution and authorization of hybrid workflows that are supported by cluster-based resource pools. Various types of authorization constraints are modeled in this paper, including role constraints, temporal constraints, cardinality constraints, Binding of Duty (BoD), Separation of Duty (SoD) constraints, and role hierarchy constraints. Workflow executions, as well as the interactions between workflow execution and authorization controls are also modeled in this paper. In this paper, the Timed Color Petri-Net (TCPN) formalism is applied to model workflow authorization and execution. Moreover, the modeling mechanism is developed in such a way that the model construction can be automated. This feature is very helpful in modeling a large collection of authorization policies or complex workflows.

The constructed models are then simulated and analyzed to obtain various performance metrics, including authorization overhead, system-oriented performance (e.g., utilization and throughput) and application-oriented performance (e.g., response time of workflows).

A high level Petri-net tool, called the CPN Tools [21,22], is utilized to implement and simulate the model. Based on the model simulations, the methods are proposed to analyze the authorization overhead and the performance bottlenecks in the system. Further, we propose the approaches to enhancing performance under the specified authorization constraints.

The work presented in this paper can be used for capacity planning, designing workload management strategies, or for estimating application performance in the presence of authorization policies. Since we can calculate from the models the overhead caused by the authorization constraints, this work also provides insight into how to tune performance by adjusting authorization policies so as to achieve a good balance between performance and security overheads.

Note that this paper investigates the executions of hybrid workflows (containing both computing tasks and human tasks) at an abstract level. Whether the execution of a hybrid workflow is semi-automated or automated is an implementation issue (depending on whether the workflow execution is programmed using BPEL4People or traditional workflow management methods), which does not affect the results obtained in this paper.

The remainder of this paper is organized as follows: Section 2 discusses related work; Section 3 introduces the Timed Color Petri-Net formalism applied in this paper; workflow authorization and execution are modeled in Section 4; model simulations and overhead analysis are discussed in Section 5. Section 5 also presents the approaches to reducing authorization overheads and improving performance. Section 6 presents the simulation results and, Section 7 concludes the paper.

2. Related work

Workflow management has been extensively studied and as a result is well documented in related literature [23,1,24,3]. Much of this research is aimed at automating the execution, and enhancing the performance, of workflows in parallel and distributed systems [1],43. Some of this research has also utilized Petri-nets to model workflow execution. However we note that their work does not formally investigate the performance of workflow execution under authorization constraints.

Research has also been conducted on the topic of security and authorization constraints in the workflow context [25,14,26–28,16]. Some of this research also uses Petri-nets to model authorization constraints. The work presented in this paper differs from this research in the following respects: First, the work found in the literature [26] does not differentiate human tasks and computing tasks, and does not model the interactions between workflow authorization and workflow execution. The work presented in this

paper models resource competition and interactions between the authorization module and the execution module. Second, the modeling approaches presented in previous literature were not developed so that the model constructions could be automated. Third, it is assumed in [25,29,14,24] that a task can only be run under one role. This assumption simplifies the modeling process. However, the assumption is not always true. It may well be the case that a task is allowed to run under a range of roles. The relaxation of this assumption is especially necessary when temporal constraints of roles' availability are taken into account. In so doing, when one role is not available, another activated role may be assigned to run the task, so that the workflow execution can still progress. In this work, the role and user assignments to tasks are modeled in a flexible fashion, which allows a task to be run under a selection of roles/users. Finally, previous work only models the execution or authorization of a single workflow [16,30,31]. The modeling mechanism developed here however, is able to model the simultaneous execution of multiple workflows.

The work in the literature [32,29,33,16,5] also investigates the workflow authorization and task delegation. However, their work focuses on guaranteeing that the authorization policies can be enforced properly when a workflow is being processed in the system. The focus of this paper is to analyze and obtain the authorization overhead as well as other performance data when running the workflows under the authorization policy. Further, this paper proposes the methods to enhance the performance, given the specified authorization constraints. This work can be used to provide support for system capacity planning or workload management and scheduling.

The Multi-layered State Machine (MLSM) is another method used in the literature [5,33] to model workflow authorization. However, the MLSM method is mainly used to guarantee that the authorization constraints are satisfied in the workflow environment, and the method itself cannot simulate and obtain the quantitative performance of the workflow execution. In order to obtain the performance data, the constructed MLSM structure needs to be converted to the Petri-nets before the performance analyses can be conducted [5,33]. Further, the work in [5,33] does not analyze the overhead caused by the authorization control, and does not investigate the methods to improve performance under authorization control, either.

In previous work [34], we have applied Generalized Stochastic Petri-Net (GSPN) theory to model workflow executions under Role-based Authorization Control, and then used standard Petri-net analysis techniques to theoretically calculate the performance from the constructed models. Although GSPN is adequate for the scenarios investigated in [34], the work did not model the workflows consisting of both human tasks and computing tasks. Also, since GSPN cannot express the temporal attributes associated with tokens, we cannot analyze the authorization overhead caused by each type of authorization policy in the models constructed using GSPN. Moreover, the work in [34] did not investigate the methods to improve the performance given the specified authorization constraints. In this paper, we will model both human and computing tasks, show how to calculate the authorization overhead in the constructed TCPN models, and based on the simulation results of the models, propose the methods to improve the performance by adjusting the scheduling strategy and the quantity of resources.

In addition, it is very difficult for the GSPN modeling scheme in [34] to achieve automated component assembly. It becomes tedious and error-prone when we need to model a large collection of authorization policies. In this work, the model is constructed in a modular fashion and individual authorization modules can be assembled automatically to form the authorization model for the entire system.

Another major difference between this work and the work in [34] is that in [34] we applied a theoretical approach to calculating the performance from the constructed models. In this paper, the developed modeling mechanism has been implemented and the performance data can be obtained by running the model simulations. These two approaches complement each other, but the simulation approach is more amenable to evaluating the irregular models (e.g., the timers associated with the timed transitions may not be exponentially distributed).

3. Timed Color Petri-Nets

The formal definition of a Color Petri-Net (CPN) differs depending on the source literature [25,35]. The CPN formalism applied in this paper is the same as that defined in [21], in which a CPN is defined as in Eq. (1).

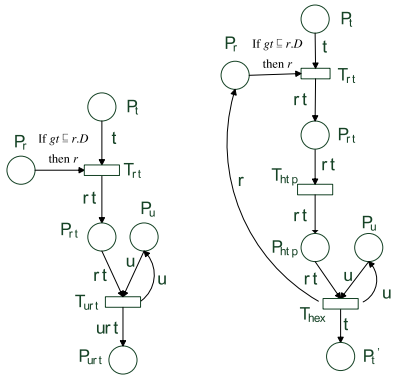
$$\text{CPN} = (P, T, AR, CS, V, CF, G, A, I) \tag{1}$$

where:

1. P is a finite set of places
2. T is a finite set of transitions such that $P \cap T = \Phi$.
3. $AR \subseteq P \times T \cup T \times P$ is a set of directed arcs.
4. CS is a finite set of non-empty color sets.
5. V is a finite set of typed variables such that $\text{Type}[v] \in CS$ for all variables $v \in V$.
6. CF: $P \rightarrow \underline{CS}$ is the color set function that assigns a color set to each place.
7. G: $T \rightarrow EXPR_V$ is a guard function that assigns a guard to each transition t such that $\text{Type}[G(t)] = \text{Bool}$, where $EXPR_V$ denotes the set of expressions over the set of variables V provided by the modeling language, Type $[e]$ denotes the type of an expression $e \in EXPR_V$, i.e., the type of the values obtained when evaluating e. The guard function defined in [21] represents the substantive difference from the CPN definitions found in other literature [25].
8. A: $AR \rightarrow EXPR_V$ is an arc expression function that assigns an arc expression to each arc ar such that $\text{Type}[A(ar)] = CF(p)_{MS}$, where p is the place connected to the arc ar and $CF(p)_{MS}$ is the set of all multisets over $CF(p)$.
9. I: $P \rightarrow EXPR_\Phi$ is an initialization function that assigns an initialization expression to each place p such that $\text{Type}[I(p)] = CF(p)_{MS}$.

A CPN model consists of places (defined in P) and transitions (defined in T), and a number of directed arcs (defined in AR). Each place can be marked with a number of tokens, and each token has a data value, which is termed the *token color*. The data value can be a primitive data type, such as an integer and a string, or a complex structure consisting of other primitive data types or complex structures. The number of tokens and the token colors in each place, called a *marking*, represents the state of the model. A place is associated with a data type (termed a *color set*), which is defined in CS. A place's color set determines the set of token colors that the tokens in the place are allowed to possess. The model determines whether a token can be fired through the arc functions associated with the arcs (defined in A) and/or guard functions associated to transitions (defined in G). An arc function evaluates to a set of tokens, which determine the type and number of tokens that can pass through the arc. An arc function or a guard function can contain a number of variables (defined in V) as well as the operations (e.g. comparison) and logical operators (e.g. if–else branch) on these variables. Therefore, an arc function or a guard function may evaluate to different values for different tokens.

A *Timed* Color Petri-Net (TCPN) is presented in [21] as an extension of a CPN. In a TCPN, a token can be associated with both a color and a time stamp. Furthermore, a CPN model has



(a) HCT. (b) HT.

Fig. 1. Role and user assignment module for HCT and HT.

a global timer, which records the current global time of the system, denoted as gt. The time stamp attached to a token t, symbolized as $t@ts$, indicates the earliest time when the token can be processed. In addition to the arc and guard function, whether a transition can fire or not in a TCPN model is also controlled by the model's global timer and the tokens' time stamps. The rule is that besides satisfying the arc and the guard functions associated with a transition, the tokens must have time stamps which are not later than gt. The model remains at a given model time as long as there are enabled transitions. If there is not such an enabled transition, the global timer is advanced to the earliest model time at which at least one transition is enabled.

The readers can refer to [21] for the detailed information of CPN and TCPN used in this paper.

4. Models

Both human tasks and human-aided computing tasks require authorization before execution. In this section, we first model various types of authorization control using TCPNs. These include: (1) Role constraints; (2) Temporal constraints; (3) Role and user assignment; (4) Binding-of-duty constraints; (5) Separation-of-duty constraints; (6) Cardinality constraints; (7) Role hierarchy constraints. We then present how to automatically assemble individual authorization modules to form the authorization control module for the entire workflow. Finally, we combine the workflow authorization module with workflow execution, and discuss the interactions between them.

4.1. Role and user assignments subject to temporal and role hierarchy constraints

As discussed in the banking example in Section 1, a HCT of a workflow involves an application being launched by a person (*user*) with a certain official position (*role*) in the bank. Generally speaking, a role and a user (e.g., an employee) need to be assigned to handle a task of a particular type in the workflow. The model of assigning a role and a user to a HCT is illustrated in Fig. 1(a). In this model, the Places P_r and P_u hold all tokens representing the roles and the users, respectively. The Pt place holds the token representing a task in the workflow. The transition T_{rt} assigns a role in the *Pr* place to a task in the *Pt* place, and deposits a token to the P_{rt} place, indicating the task-role assignment has been established. When a token is deposited into the place P_{urt}, it means that a user has been assigned to the task. After the HCT has completed authorization, it is sent to the computing resource pool for execution (the execution of a HCT is modeled in Section 4.4). Since the time spent by a user (human resource) to handle a HCT is negligible, the user assigned to the HCT is free to handle other tasks. Therefore, the user token is returned to the place P_u immediately after T_{urt} fires.

The model for conducting the role and user assignment for an HT is shown in Fig. 1(b). A role is first assigned to the HT by the transition T_{rt}. Since an HT is processed by human resources, its user assignment procedure and execution procedure are combined together. After the HT completes the role assignment, it is deposited by the transition T_{htp} into a Human Task Pool (the place P_{htp}), where the HT waits to be allocated to a user. Note that there is only one P_{htp} place in the system model, and all HTs that have completed the role assignment will be put into the place. The guard function of the transition T_{hex} is used to enforce the scheduling strategy, which will be discussed in Subsection IV.D. After the HT finishes execution, the role and the user are returned to their corresponding places and are free to be allocated to other tasks. Moreover a token is deposited to the place (P_t' in Fig. 1(b)) corresponding to the child task of task t.

We call the model of assigning a role and a user to a task a *role and user assignment module*. A CPN model is formally defined using Eq. (1). The remainder of this subsection is dedicated to determining the attributes CS, CF, A and G (i.e., token colors, arc and Guard functions) for the role and user assignment module in Fig. 1. These attributes together enforce the temporal constraints and role constraints. It is straightforward to determine the attributes P, T, AR, V and I (i.e., the initial number of tokens in the places) in the model. Therefore, they are omitted for brevity. After determining these attributes, the role and user assignment module can be automatically assembled to construct the authorization control module for the entire workflow, which will be shown in Section 4.3.

(1) *Token colors.*

- A token in the *Pr* place, denoted as r, represents a role. The color of the token r is defined as $r = (rid, D)$, where rid is the role identifier; D defines the temporal attribute of the role and it is a set of durations in which the role is activated and can be assigned to tasks. D can be expressed as in Eq. (2), where $\aleph$ is the set of natural numbers.

$$D = \{[ld_i, ud_i] | i \in \aleph\}. \tag{2}$$

- A token in the *Pt* place, denoted as t, represents a task. The token color is defined as $t = (tid, wid, iid, e)$, where tid is the task identifier, wid is the identifier of the workflow that the task belongs to, iid is the identifier of the workflow instance, and e is the task's execution time. A time stamp ts is attached to the token t, denoted as $t@ts$. The time stamp represents the earliest time at which the task can be processed. The time stamp of the token t is set as the time when the token is deposited to Place P_t.

- The color of a token in the P_{rt} place is defined as follows, which is the combination of the color attributes of token t and r.

  $rt = (tid, wid, iid, e, rid, D)$

- A token in place Pu represents a user, whose color is defined as $u = (uid, rid)$, where rid is the role that the user belongs to.
- The color of a token in the place P_{urt} in Fig. 1(a) is defined as

  $urt = (uid, rid, tid, wid, iid, e, D)$.

- A token in Place P_{hex} in Fig. 1(b) is the same as the token in Place P_{rt}.

(2) *Arc functions.*

- The arc function (APr, T_{rt}) can be defined as "If $gt \sqsubseteq r.D$ then r", which is used to enforce the roles' *temporal constraints* (gt is the current global time of the system). This expression means a role is allowed to be assigned to a task only when the value of the global timer is within one of the role's activation durations (denoted by the symbol "$\sqsubseteq$"), i.e., the role is available. It is assumed in this paper that the users have the same availability period as their associated roles. But it is straightforward to extend the model to allow individual users to have different and more restricted availability (e.g., in the case of different employees rotating to cover different shifts of the same official position).
- Other arc functions in Fig. 1(a) are defined as $A(P_t, T_{rt}) =$ "t", $A(Pu, T_{urt}) =$ "u", $A(P_{rt}, T_{urt}) =$ "rt" and $A(T_{urt}, P_{urt}) =$ "urt". In Fig. 1(b), $A(P_{rt}, T_{htp}) =$ "rt", $A(T_{htp}, P_{htp}) =$ "rt", $A(P_{htp}, T_{hex}) =$ "rt", $A(P_u, T_{hex}) =$ "u", $A(T_{hex}, P_r) =$ "r", $A(T_{hex}, P_u) =$ "u".

(3) *Guard functions.*

The guard function associated to transition T_{rt}, denoted as (GT_{rt}), is used to enforce role assignment restrictions. Three types of restriction are modeled. The first one specifies that a role is assigned to a task only when the task can run to completion within one of the role's activation durations specified in Eq. (2). This restriction is motivated by the fact that most existing workflow description languages, such as BPMN [35], assume that the execution of a task in a workflow is an atomic process. This restriction is formulated as

$$[gt, gt + t \cdot e] \sqsubseteq r \cdot D. \tag{3}$$

The second type of restriction is the *role constraint* which specifies the set of roles that are allowed to run a task. This restriction is expressed as

$$r \cdot rid \in R(t), \tag{4}$$

where $R(t)$ denotes the set of roles which can run the task t.

The third type of restriction reflects the *role hierarchy constraint*. The role hierarchy constraint ensures that in all available roles, only the role with the least privilege is assigned to run a task. Eqs. (3) and (4) can be used to determine which roles are available to run a task. The available roles in $R(t_i)$ can be divided into the disjoint sets, $s_1, s_2, \ldots, s_h$, based on the role hierarchy relation, i.e., the roles having the hierarchy relation are classified into the same set. Then the role hierarchy constraint can be formalized as follows, where the operator *min* returns the role with the least privilege in the set s_i.

$$r \cdot rid \in \{r_j \cdot rid | r_j = \min(s_i), 1 \le i \le h\}. \tag{5}$$

Note that the role hierarchy constraint is a special case of the role constraint, since s_i in Eq. (5) is a subset of $R(t_i)$. This means that if a task-role assignment satisfies the role hierarchy constraint, it must also satisfy the role constraint. Therefore, (GT_{rt}) can be expressed as in Eq. (6).

$$[gt, gt + t \cdot e] \sqsubseteq r \cdot D \&\& r \cdot rid \in \{r_j \cdot rid | r_j = \min(s_i), 1 \le i \le h\}. \tag{6}$$

In Fig. 1(a), the guard function of Transition T_{urt}, $G(T_{urt})$, guarantees that only the users which belong to the role assigned to the task are eligible for the task-user assignment. $U(rid)$ denotes the set of users belonging to the role whose id is rid. $G(T_{urt})$ can be formalized as

$$G(T_{urt}) = \text{"}u \cdot uid \in U(rt \cdot rid)\text{"}. \tag{7}$$

Since the user assignment and the task execution for HTs are combined in the same module, the guard function of Transition T_{hex}, $G(T_{hex})$, is used not only to conduct user assignment, but also to enforce the scheduling strategy. The expression used to perform the user assignment in $G(T_{hex})$ is the same as Eq. (7). The expression for enforcing the scheduling strategy is discussed in Section 4.4.

4.2. Binding of duty and separation of duty constraints

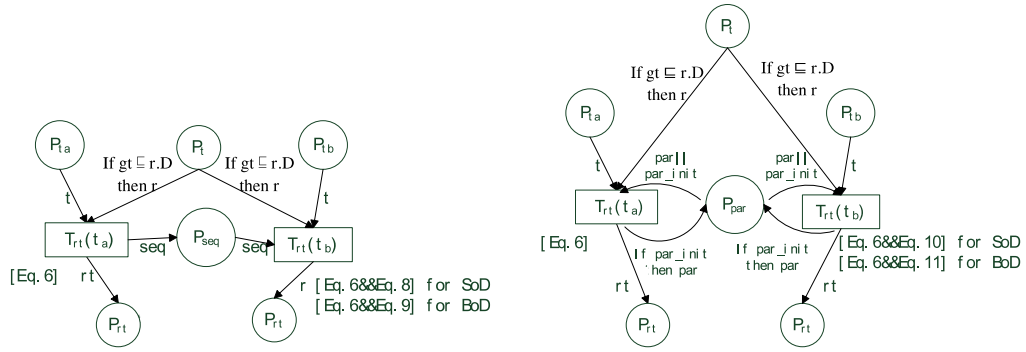
The duty constraints impose restrictions on the role or user assignments of two tasks in a workflow. The paper only presents the duty constraints for roles. The duty constraints for users can be modeled in the similar fashion. Although Binding of Duty (BoD) and Separation of Duty (SoD) represent opposite authorization behaviors, these two types of duty constraints are modeled in a uniform fashion in this paper so that they have the same model structure. We call these the SoD module and the BoD module. The differences between these are essentially in some arc and guard functions, as well as the color set of some places. The benefit of doing this is that the models for individual duty constraints can be easily assembled to form the authorization model for the entire workflow.

There are two types of relationship between two tasks in a workflow in terms of precedence constraints (i.e., the order of execution): sequential tasks and parallel tasks. Assume task ta and tb. If $tb \in$ Pred (ta) or $tb \in$ Succ (ta) (Pred (ti) and Succ (ti) denote the set of tasks which are task ti's predecessors and successors, respectively), then ta and tb are sequential tasks and have to be run in the required order. If $tb \notin$ Pred (ta) and $tb \notin$ Succ (ta), ta and tb can be executed in parallel. Duty constraints are modeled in a different way for these two types of tasks. Their model structures are shown in Fig. 2(a) and (b), respectively.

(1) *Sequential tasks.*

As shown in Fig. 2(a), the duty constraints module consists of the role assignment modules for ta and tb (encompassed in two round-cornered rectangles), connected by place P_{seq}. P_{seq} is one of the output places of transition T_{rt} (ta) and one of the input places of transition T_{rt} (tb). The attributes of the role assignment modules have been discussed in Section 4.1. The attributes related to the new place, P_{seq}, are as follows.

- **Token colors**: The color of a token in place P_{seq} is defined as $seq = (tid, wid, iid, rid)$, which carries the information of which role has been assigned to task tid.
- **Arc functions**: $(AT_{rt}(ta), P_{seq})$ and $A(P_{seq}, T_{rt}(tb))$ are both defined as "seq".
- **Guard functions**: The guard functions associated to T_{rt} (tb) are different for SoD and BoD constraints. A SoD constraint can be expressed as Eq. (8), while a BoD constraint can be formulated as Eq. (9). t in Eqs. (8) and (9) is a token in place P_{tb}. The condition $seq.wid = t.wid$and $seq.iid = t.iid$ is used to guarantee that the same workflow instance is referred to, since this model allows multiple instances of the same workflow to be processed simultaneously. Different instances of a workflow



(a) Sequential tasks. (b) Parallel tasks.

Fig. 2. Duty constraints modules.

will have different values of *iid*. The guard function $G(T_{rt}(tb))$ can be formulated as Eqs. (6) and (8) for the SoD constraint, and as Eqs. (6) and (9) for BoD.

$$\begin{aligned} seq \cdot wid &= t \cdot wid \&\& seq \cdot iid \\ &= t \cdot iid \&\& seq \cdot rid \neq r \cdot rid \end{aligned} \tag{8}$$

$$\begin{aligned} seq \cdot wid &= t \cdot wid \&\& seq \cdot iid = t \cdot iid \&\& seq \cdot rid \\ &= r \cdot rid. \end{aligned} \tag{9}$$

(2) *Parallel tasks.*

Similar to Fig. 2(a), a new place, labeled P_{par}, is used in Fig. 2(b) to interface between the role and user assignment modules when t_a and t_b are parallel tasks. In the remainder of this subsection we first determine the attributes related to Place P_{par}, and then use an example to illustrate the workings of the module.

- **Token colors**: There are two types of tokens in P_{par}: *par_init* and *par*. *par_init* is defined as $par\_init = (tid_a, tid_b, wid, iid, flag)$, while *par* is defined as $par = (tid, rid, wid, iid, flag)$, where the fifth parameter *flag* is used to differentiate the tokens *par_init* and *par*. The value of *flag* is set to be zero for *par_init* while its value be 1 for *par*. In doing so, the model can differentiate *par_init* and *par* by checking the value of *flag*.
- **Guard functions**: If it is a SoD constraint, the guard function of the transition T_{rt} (ta) (or $T_{rt}(t_b)$) is formulated as Eqs. (6) and (10). If it is a BoD constraint, it is expressed as Eqs. (6) and (11).

$$\begin{aligned} &((t \cdot tid = par\_init \cdot tida \parallel t \cdot tid = par\_init \cdot tidb) \&\& \\ &\quad t \cdot wid = par\_init \cdot wid \&\& par\_init \cdot iid = t \cdot iid) \parallel \\ &\quad (t \cdot wid = par \cdot wid \&\& par \cdot iid = t \cdot iid \&\& \\ &\quad r \cdot rid \neq par \cdot rid) \end{aligned} \tag{10}$$

$$\begin{aligned} &((t \cdot tid = par\_init \cdot tida \parallel t \cdot tid = par\_init \cdot tid) \&\& \\ &\quad t \cdot wid = par\_init \cdot wid \&\& par\_init \cdot iid = t \cdot iid) \parallel \\ &\quad (t \cdot wid = par \cdot wid \&\& par \cdot iid = t \cdot iid \&\& \\ &\quad r \cdot rid = par \cdot rid). \end{aligned} \tag{11}$$

- **Arc functions**: $A(P_{par}, T_{rt})$ and $A(T_{rt}, P_{par})$ are defined in Eqs. (12) and (13), respectively.

$$A(P\_par, T\_rt) = par\_init \parallel par \tag{12}$$

$$A(T\_rt, P\_par) = \text{“}\textbf{if}\ par\_init\ \textbf{then}\ par\text{”}. \tag{13}$$

(3) *Workings of the duty constraint modules.*

The workings of the duty constraint modules and the above expressions are illustrated as follows. When performing the role assignment for a task (e.g., *ta*), the model will check whether the other task (e.g., *tb*) has been assigned a role. Assume there is a BoD constraint between *ta* and *tb*, then the place P_{par} will contain a *par_init* token in the model's initial marking. When the T_{rt} transition performs the role assignment for *ta*, it will evaluate which token in P_{par} can satisfy Eq. (10), and therefore can be fired by the transition. There are two possibilities:

(a) If there is a corresponding *par_init* token in P_{par}, which means that *tb* has not been assigned a role, then the first part of Eq. (10) (i.e., the portion before the second "||") will be evaluated as true. Consequently, the T_{rt} (ta) transition will remove the *par_init* token from P_{par} and deposit a *par* token back to P_{par} as shown in Eq. (13).

(b) If there is a *par* token in P_{par}, this means that *tb* has been assigned to a role. Further, if there is a role in place P_r whose identifier (i.e., *r.rid*) is the same as the role assigned to *tb* (i.e., *par.rid*), the second part of Eq. (10) will be evaluated as true. Thus, the BoD constraint is enforced.

4.3. Assembling authorization modules

One of the advantages of the modeling mechanism developed in this paper is that the authorization model can be constructed automatically by assembling a set of interacting hierarchical modules. There are clear interfaces and hierarchy structure among the modules. As shown in Fig. 2, the duty constraints module consists of the role modules for *ta* and *tb*, interfacing via place P_{par} or P_{seq}. Generally, the assembly procedure is as follows.

Definition 1. $M_k = (P_k, T_k, AR_k, CS_k, V_k, CF_k, G_k, A_k, I_k)$ is the role and user assignment module for task *tk*.

Definition 2. $M = (P, T, AR, CS, V, CF, G, A, I)$ is a module in which j tasks, $t_{i1}, t_{i2}, \ldots, t_{ij}$, have the duty constraints with task *tk*.

Definition 3. $M' = (P', T', AR', CS', V', CF', G', A', I')$ is the module that captures the duty constraints between the j tasks in module M and task t_k in module M_k.

Then, the module M' in Definition 3 can be constructed by assembling M and M_k as shown in Theorem 1.

Theorem 1. *Given M_k, M, and M' in Definitions 1–3, the attributes of M' can be computed as follows.*

- $$P' = P \cup P_k \cup \left\{ \bigcup_{t_{ik} \in Pred(t_k) \cup Succ(t_k)} P_{seq}(t_{i_k}, t_k) \right\} \cup \left\{ \bigcup_{t_{i_k} \notin Pred(t_k)} P_{par}(t_{i_k}, t_k) \right\}. \tag{14}$$

- $$T' = T \cup Tk \tag{15}$$
- $$CS' = CS \cup CSk \cup seq \cup \{par, par\_init\} \tag{16}$$
- $$CF' = CF \cup CF_k \cup CF(P_{seq}) \cup CF(P_{par}). \tag{17}$$
- *A' can be computed using Eq. (18), where $P_{seq}(t_x, t_y)$ or $P_{par}(t_x, t_y)$ denotes the P_{seq} or P_{par} place that is added to model the duty constraints between t_x and t_y.*
- *AR' and G' can be computed using Algorithm 1, where g^{seq}_{SoD}, g^{seq}_{BoD}, g^{par}_{SoD} and g^{par}_{BoD} are the guard expressions specified in Eqs. (8)–(11), respectively.*

Proof. (1) For each task in the module M that has the duty constraint with task t_k, a new place (either P_{seq} or P_{par} depending on whether they have the precedence constraint) is added in the new module M'. So P' can be computed as in Eq. (14). (2) There is no need to add new transitions in M'. So T' can be computed as Eq. (15). (3) The new color set of M' is the union of the color sets of M_k and M plus the colors of the tokens in Place P_{seq} and P_{par}. So CS' can be computed as Eq. (16). Similarly, CF' can be computed using Eq. (17). (4) The arc function set in M' is the union of the arc function sets of M_k and M plus the arcs added between place P_{seq} or P_{par} and the corresponding T_{rt} transitions, as shown in Fig. 2. So A' can be computed using Eq. (18). (5) Since there are no new transitions added in M', Step 1 to 3 in Algorithm 1 first initializes AR' to be the union of $AR$and AR_k, and G' be the union of G and G_k. Then, in the for-loop, the algorithm adjusts the input, output and guard functions of the individual transitions that have the arc connections with the newly added P_{seq} and P_{par} places. □

The importance of Theorem 1 is that with the formalized equations and algorithm, the process of constructing the authorization control model can be automated rather than being built manually, which can greatly speedup the modeling process.

We use a case study to illustrate how to assemble the individual role assignment modules subject to the duty constraints. The exemplar workflow is abstracted from a business process in a bank [36]. The roles which are involved in the process are listed in Table 1. The workflow consists of 7 tasks, whose topology is shown in Fig. 3(a). It is assumed that task t_1 is an automated computing task and does not need the role and user assignment. The tasks and the role assignment constraints are shown in Table 2.

Assume that the following BoD and SoD constraints are imposed on the tasks, where $r(t_i)$ denotes the role assigned to task t_i.

$C1 : r(t2) = r(t4);$ $\quad C2 : r(t2) \neq r(t5);$

$C3 : r(t2) \neq r(t7);$

$C4 : r(t6) \neq r(t7);$ $\quad C5 : r(t3) = r(t5).$

These duty constraints can be represented as a *duty constraints graph* shown in Fig. 3(b). In the duty constraints graph, if there are duty constraints between two sequential tasks, a single-headed arrow is used to connect the predecessor to the successor. If there exist duty constraints between two parallel tasks, the two tasks are connected by a double-headed arrow. Applying the module assembly operations described in Eqs. (14)–(18) and Algorithm 1, the hierarchy of the authorization control model for the entire workflow can be constructed as shown in Fig. 4, where M_{urti} (i.e., a round-cornered rectangle) is a role and user assignment module for task t_i if t_i is an HCT, or is a role assignment module if t_i is an HT. Note that there should be a directed arc from the *Pr* place and the *Pu* place to every module as shown in Fig. 2, these are omitted for clarity.

Table 1
Role descriptions in the loan lending workflow.

Role	Description	Role	Description
SM	Second market official	BM	Bank manager
FA	Financial advisor	CL	Bank clerk
LB	Loan broker	UW	Underwriter

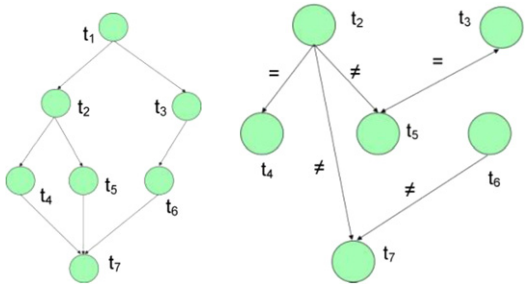
Table 2
Task descriptions and role constraints in the workflow.

Task	Description	Role constraints
t_1	Updating products and rates	None
t_2	Product and rate decision engine	FA/LB/BM
t_3	Collecting data	FA/LB/CL
t_4	Analyzing data of third-party 1	FA/LB/CL
t_5	Analyzing data of third-party 2	FA/LB/CL
t_6	Analyzing business rules	FA/BM
T_7	Underwriting	UW/BM

Algorithm 1. Calculating AR' and G' from M and M_k.

1. $AR' = AR \cup AR_k$
2. $G' = G \cup G_k$
3. for each task t_{i_s} in M that has BoD or SoD with t_k do
4. If $t_{i_s} \in Pred(t_k)$ then {
5. $AR' = AR' \cup \{(T_{rt}(t_k), P_{seq}(t_{is}, t_k)), (P_{seq}(t_{is}, t_k), T_{rt}(t_{is}))\}$
6. if there is SoD between t_k and t_{is} then
7. $G'(T_{rt}(t_k)) = G_k(T_{rt}(t_k)) \&\& g^{seq}_{SoD}$
8. else
9. $G'(T_{rt}(t_k)) = G_k(T_{rt}(t_k)) \&\& g^{seq}_{BoD}$
10. }
11. else if $t_{is} \in Succ(t_k)$ then {
12. $AR' = AR' \cup \{(T_{rt}(t_{is}), P_{seq}(t_{is}, t_k)), (P_{seq}(t_{is}, t_k), T_{rt}(t_k))\}$
13. if there is SoD between t_k and t_{is} then
14. $G'(T_{rt}(t_{is})) = G(T_{rt}(t_{is})) \&\& g^{seq}_{SoD}$
15. else
16. $G'(T_{rt}(t_{is})) = G(T_{rt}(t_{is})) \&\& g^{seq}_{BoD}$
17. }
18. else { // t_{is} and t_k can be run in parallel
19. $AR' = AR' \cup \{(P_{par}(t_{is}, t_k), T_{rt}(t_{is})), (P_{par}(t_{is}, t_k), T_{rt}(t_k)), (T_{rt}(t_{is}), P_{par}(t_{is}, t_k)), (T_{rt}(t_k), P_{par}(t_{is}, t_k))\}$
20. if there is SoD between t_k and t_{is} then
21. $G'(T_{rt}(t_k)) = G_k(T_{rt}(t_k)) \&\& g^{par}_{SoD}$
22. $G'(T_{rt}(t_{is})) = G(T_{rt}(t_{is})) \&\& g^{par}_{SoD}$
23. else
24. $G'(T_{rt}(t_k)) = G_k(T_{rt}(t_k)) \&\& g^{par}_{BoD}$
25. $G'(T_{rt}(t_{is})) = G(T_{rt}(t_{is})) \&\& g^{par}_{BoD}$
26. }

$$\begin{aligned} A' = {} & A \cup A_k \cup \bigcup_{t_{is} \in Pred(t_k)} A(T_{rt}(t_{is}), P_{seq}(t_{is}, t_k)) \\ & \cup \bigcup_{t_{is} \in Succ(t_k)} A(P_{seq}(t_{is}, t_k), T_{rt}(t_{is})) \\ & \cup \bigcup_{t_{is} \notin Pred(t_k) \cup Succ(t_k)} (A(P_{par}(t_{is}, t_k), T_{rt}(t_{is})) \\ & \cup A(P_{par}(t_{is}, t_k), T_{rt}(t_k)) \\ & \cup A(T_{rt}(t_{is}), P_{par}(t_{is}, t_k)) \cup A(T_{rt}(t_k), P_{par}(t_{is}, t_k))). \end{aligned} \tag{18}$$



(a) An exemplar workflow. (b) Duty constraints graph.

Fig. 3. An exemplar workflow and its duty constraints graph.

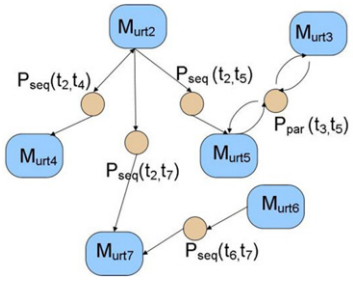


Fig. 4. Hierarchy of the authorization control module for the workflow in Fig. 3(a).

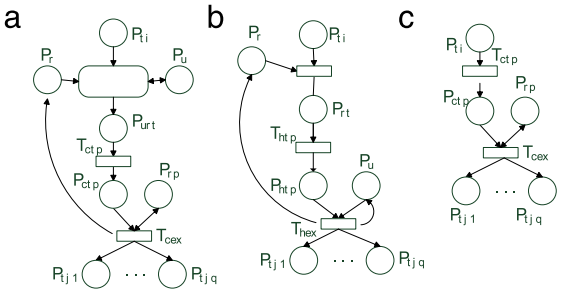


Fig. 5. Task execution module under authorization control (a) for human-aided computing task (HCT); (b) for human task (HT); (c) execution module for automated computing task (ACT).

In this work we model the fact that the tokens in *Pr* and *Pu* are shared by all tasks. This is reasonable because the roles and users are the global parameters and should be applied to all tasks in the system. *Cardinality constraints*, which specify the maximum number of tasks that can be handled at the same time by a role, can be modeled by the number of tokens representing the role in *Pr*.

Since the users are human resources and a user can only process one task at a time, in the P_u place the number of the token corresponding to a user is one. Note that when authorizing the HCTs in Fig. 1(a), the user token is returned to the P_u place immediately after the authorization is completed (we assume that the time spent by a user to handle a HCT is negligible). Therefore, there may be multiple tasks running under a user in the system. This paper does not assume the explicit cardinality constraint for users. The cardinality constraint of the users is implicitly reflected by that of the role that the users are affiliated with.

4.4. Modeling workflow execution under authorization

This subsection first presents the modules for executing a HCT, an HT and an ACT, which are shown in Fig. 5(a)–(c), respectively, and then discusses the model for executing a workflow under the authorization control as well as the interactions between workflow authorization and workflow execution.

4.4.1. HCT and ACT executions under authorization

Fig. 5(a) shows the execution of a HCT. A HCT requires the role and user assignment, so it is necessary to go through the role and user assignment module as shown in Fig. 1(a); this is abstracted here by a round-cornered rectangle with P_t, P_r and P_u as the input places of the module and P_{urt} as the output place. The place P_{ctp} is a computing ask pool, which is used to hold all computing tasks (i.e., ACTs and HCTs). After the task is assigned a role and a user (i.e., a token is deposited into the P_{urt} place), it is put into the P_{ctp} place. The token in Place P_{ctp} is the same as that in P_{urt}. Note that there is only one P_{ctp} place in the model, which is used to hold all HCTs and ACTs ready for execution.

The place P_{rp} in the figure represents the computing resource pool. In place P_{rp}, a token represents a computing resource, and the number of the tokens represents the number of resources currently available in the pool. A token in the P_{rp} place is defined as $rp = nid$.

The T_{cex} transition controls task scheduling and resource allocation, and also represents task execution.

The scheduling of computing tasks is controlled by the guard functions of the transition T_{cex} (i.e., $G(T_{cex})$), respectively. If a First-Come-First-Served (FCFS) policy is used to schedule the tasks, $G(T_{cex})$ can be defined as Eq. (19), where τ denotes a token in the P_{ctp} place, i.e., *urt*. The expression uses the tokens' time stamps to determine which token first arrives at the P_{ctp} place.

$$\tau \in \{\tau | \tau@ts = \min\{\tau@ts\}\}. \tag{19}$$

During the resource allocation, assume an authorized task is randomly allocated to any of the free computing resources in the pool. Resource allocation and task execution are modeled in the following way. If there are tokens in the P_{rp} place (i.e., there are free computing resources) and there is at least a token in P_{ctp}, the T_{cex} transition fires immediately (which indicates the start of the task's execution and which task starts execution is controlled by Eq. (19)).

According to the workflow topology, only when a task is completed, can the task's children start execution. Assuming $w_k.t_i$ is the currently running task in workflow w_k and $Children\ (w_k.t_i) = \{t_{j1}, t_{j2}, \ldots t_{jq}\}$ denotes the set of tasks which are t_i's children in workflow w_k. Then when T_{cex} fires, the token $t = (tid_{jk}, wid, iid, e)$ is deposited into the place corresponding to task t_{jk}, i.e., Place P_{tjk}. The time stamp of the deposited token t will be set as the current global time plus t_i's execution time, that is, $t@ts = gt + t_i.e$. Therefore, task t_{jk} is not ready for authorization and execution until task t_i is completed.

The arc function $A(T_{cex}, P_{tjk})$ is used to control which places a token should be deposited into. $A(T_{cex}, P_{tjk})$ can be defined as

$$A(T_{cex}, P_{tjk}) = \text{“if } t_{jk} \in Children(w_k.t_i),\ \text{then } t_{jk}\text{”}. \tag{20}$$

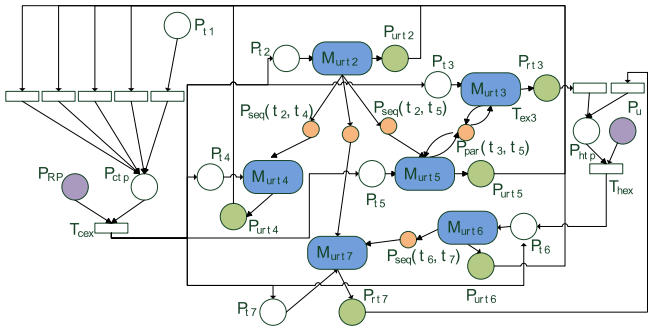


Fig. 6. Modeling workflow execution under authorization control; only the Petri-net components for task t_1 are labeled, and the labels for other tasks are omitted for the sake of clarity); the place *Prp* represents the node pool and the individual *Prp* places should be regarded as a single *Prp* place.

Moreover, after a task is completed, the role assigned to the task needs to be returned to place *Pr* (the user token has been returned to the P_u place after T_{urt} fires as shown in Fig. 1(a)), so that the role can be assigned to other tasks and its cardinality constraint can be applied. This procedure is modeled as follows. When T_{cex} fires, an *r* token is deposited back to the *Pr* place. The time stamp of the token is set as $gt + t \cdot e$. Similarly, when T_{cex} fires, an *rp*token is deposited back to the P_{rp} place (expressed as double-headed arrows). *rp@ts*is also set as $gt + t \cdot e$, which means although the *rp* token is back to the P_{rp} place, the corresponding resource can only be allocated to other tasks when the simulated execution of the current task has been completed.

Fig. 5(c) shows the execution module for an Automated Computing Task (ACT). An ACT does not need the role and user assignment. Therefore, it is put into the P_{ctp} place immediately. Afterward, it is handled in the similar way as the HCTs. Namely, when there are tokens in P_{rp}, the ACT can start execution. After the task is completed, the new tokens are deposited into the places corresponding to the task's children.

4.4.2. HT execution under authorization

Fig. 5(b) shows the HT execution under the authorization control. After an HT is assigned a role, it is put into a Human Task Pool, Place P_{htp}, where it waits for user assignment and execution. The guard function $G(T_{hex})$ enforces the scheduling strategy and user assignment. If the FCFS scheduling is applied, the expression for enforcing the scheduling strategy is defined in Eq. (19). The only difference is that τ is now an *rt*, instead of an *urt*. Therefore, combined with the expression for performing the user assignment shown in (7), $G(T_{hex})$ can be defined as (21), where τ is the *rt* token.

$$G(T_{hex}) = u \cdot uid \in U(rt \cdot rid)\&\& \\ \tau \in \{\tau | \tau@ts = \min\{\tau@ts\}\}. \quad (21)$$

The arc function $A(T_{hex}, P_{tjk})$ is the same as Eq. (20).

When T_{hex} fires, a *u* token is deposited back to the P_u place and the time stamp of the *u* token is set as $gt + t \cdot e$, which means that the user is not available to handle other tasks until the current task is completed.

4.4.3. Workflow execution under authorization

Fig. 6 models the execution of the workflow in Fig. 3(a) under authorization control. Assume that task t_1 is an ACT, t_3 and t_7 are the HTs and the remaining tasks are the HCTs. Similar as in Fig. 4, the role and user assignment module for task t_i is represented as a round-cornered rectangle (labeled as M_{urti}). The role and user assignment module of HCTs has an input place P_{ti} and an output place P_{urti}, while the module for HTs has an input place P_{ti} and an output place P_{rti}. Note that in this figure, there is only one P_{ctp} place,

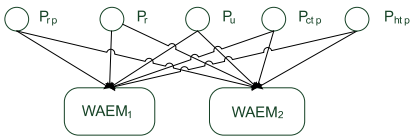


Fig. 7. Authorization and executions of multiple workflows (submitted by different clients); WAEM_i is the workflow authorization and execution module for workflow w_i.

which holds all authorized computing tasks, and one P_{htp} place, which holds all human tasks.

The workings in the model are explained as follows. Since t_1 is an ACT, it does not need authorization and it is sent to the P_{ctp} place immediately. After t_1 is run (i.e., the transition T_{cex} fires), a token is deposited into the place t_2 and t_3, according to the arc function $A(T_{cex}, P_{ti})$ $(2 \leq i \leq 7)$, defined in Eq. (20). This starts the authorization process for t_2 and t_3. After t_2's authorization is completed, an *urt* token is deposited into the places P_{urt2}, and the *urt* token is further deposited into the place P_{ctp}, where the authorized t_2 waits to be allocated a computing resource and then to be executed. T_3 is an HT. After a role is assigned to t_3, it is put into the P_{htp} place, where it waits to be scheduled to a user for execution. The scheduling strategy is enforced by Eq. (21). The authorization and execution of task $t_4 - t_7$ follow the same procedure.

It can be seen from Fig. 6 that there is a clear interface between the workflow authorization module and the task execution module. The P_{ti} places are the input interface of the workflow authorization module and the output interface of task executions, while the P_{ctp} and P_{htp} places are the output interface of the workflow authorization module and the input interface of the task executions.

Hierarchical Colored Petri Nets are defined in [21], in which a Petri Net model can have arbitrary number of abstraction levels. Each abstraction level contains the places, transitions and *substitution transitions*. A substitution transition is mapped to a module. The detailed structure of a module can be established in the lower abstraction levels [21]. Note that although the term "module" is introduced in this paper, for example, "role and user assignment module", "workflow authorization module", etc., the modules are introduced only for the purpose of understanding the conceptual hierarchy of the constructed model. These modules do not precisely comply with the concept of *module* and *substitution transition* defined in [21]. Therefore, the entire model constructed in this paper is a non-hierarchical model.

We found that it is difficult to use the Hierarchical Colored Petri Nets defined in [21] to model the workflow authorization and execution system in this paper. The reason is explained as follows. In the Hierarchical Colored Petri Nets defined in [21], a module should have input and output ports, and the modules are tied together through their input and output ports to form the higher level module. In the workflow authorization module in this paper, however, the role and user assignment module are combined together by adding new places (P_{seq} or P_{par}) to form the workflow authorization system. Moreover, the net structure of the workflow authorization system varies with the workflow topology and the BoD and SoD constraints present among the tasks in the workflow.

If multiple workflows are running in the system (each workflow can have many instances), an authorization and execution module shown in Fig. 6 needs to be constructed for each workflow. Fig. 7 shows the model structure for authorizing and executing two workflows. In the figure, WAEM_1 and WAEM_2 are the authorization and execution modules for workflow w_1 and w_2, respectively. As can be seen from the figure, the different workflow authorization and execution modules only share Place P_{RP}, P_r, P_u, P_{htp} and P_{ctp}.

5. Model simulation and analysis

The modeling mechanism presented in this paper has been implemented using the CPN Tools [22]. The CPN Tools is a software platform that is able to construct and simulate the Petri-net models. This toolkit provides a flexible mechanism that allows users to monitor a set of tokens, places and/or transitions, and their runtime status can be automatically collected during model simulations. This functionality of the toolkit is utilized in this paper to analyze and evaluate a constructed Petri-net model in terms of authorization overhead and various performance metrics.

5.1. Analyzing overhead caused by authorization

Analyzing authorization overhead in a constructed model can give us insight as to which authorization requirements are causing performance degradation under the current authorization deployment. The authorization overhead can be represented by the time a task has to wait for accomplishing the role and user assignment. In this paper, it is assumed that the roles and their associated users have the same temporal constraints, and that there are no separate cardinality constraints applied to the users. Consequently, once the task-role assignment is completed, there should be no further waiting for the task-user assignment for HCTs. For HTs, if a task has to wait for users, the waiting is caused by resource competition (i.e., all required users are busy for processing other HTs), and should not be regarded the authorization overhead. Therefore, a task's authorization overhead is equal to the time for assigning roles. It is easy to extend this work to incorporate separate temporal constraints and cardinality constraints for users.

5.1.1. Overhead by cardinality and temporal constraints

Fig. 1 shows the role and user assignment module, which is the building block of the authorization module for the entire workflow. During the model simulations, the CPN Tools can extract the time stamp of a token in Place P_t (i.e., the arrival time of the corresponding task) and the time stamp of a token in Place P_{rt} when the task has been assigned a role. The difference between these two time stamps is the time the task has to wait for the role assignment.

$w(t_j)$ denotes the waiting time for accomplishing the role assignment for task t_j. The waiting is caused by either cardinality constraint or temporal constraint. $w(t_j)$ can be calculated as,

$$w(t_j) = rt_j@ts - t_j@ts. \tag{22}$$

$w(t_j, r_i)$ denotes the authorization waiting time if role r_i ($r_i \in R(t_j)$) is assigned to task t_j. If r_i is activated, $w(t_j, r_i)$ represents the authorization overhead caused by r_i's cardinality constraint to t_j (which we term as r_i's cardinality overhead on t_j). In this case, $w(t_j, r_i)$ can be calculated using Eq. (23), where r_i^e denotes the instance of r_i that has the earliest timestamp when the token is deposited into Place P_{ti}; r_i^e @ts and $t_j@ts$ denote the time stamp of role instance r_i^e and that of task t_j, respectively. The setting of a role instance's time stamp is discussed in the second last paragraph of Section 4.4.1.

$$w(t_j, r_i) = \max(r_i^e@ts - t_j@ts, 0). \tag{23}$$

If r_i is not activated, then $w(t_j, r_i)$ represents the overhead caused by r_i's temporal constraint. In this case, $w(t_j, r_i)$ can be calculated using Eq. (24), where $ld^e(r_i)$ is the earliest time when r_i becomes activated. $ld^e(r_i)$ can be determined by D (the activated durations of r_i) defined in Eq. (2).

$$w(t_j, r_i) = \begin{cases} 0 & [gt, gt + t_j \cdot e] \subseteq r_i \cdot D \\ ld^e(r_i) - TS(t_j) & \text{otherwise.} \end{cases} \tag{24}$$

Using Eqs. (23) and (24), we can calculate every task's authorization overhead caused by every role's cardinality constraint and temporal constraint. Note that the following relation holds.

$$w(t_j) = \min\{w(t_j, r_i) \mid r_i \in R(t_j)\}.$$

5.1.2. Overhead by role, duty and hierarchy constraints

Role constraints, role hierarchy constraints and duty constraints have the similar impact, i.e., limiting the range of roles eligible for role assignment.

The overhead caused by these three constraints to task t_j is represented in the following way. We first calculate the waiting time for accomplishing the role assignment for t_j (i.e., $w(t_j)$), then calculate t_j's authorization waiting time disregarding the constraint. The difference between these two authorization waiting times is deemed the overhead caused by the constraint to task t_i. The average of the overhead over all tasks is regarded as the overhead of the constraint. The following discusses how to disregard these constraints in the model.

(i) *Disregarding role constraints.*

The role constraints are reflected in Eq. (6) (note that the role constraint is modeled as a special case of the role hierarchy constraint). When the role constraint for task t_i is disregarded, the entire role set has to be examined when enforcing the role hierarchy constraint (not just considering the roles in $R(t_i)$). Assume all roles in the system can be classified into Q disjoint sets, $H_1, H_2, \ldots, H_Q$, based on the role hierarchy relation. $H(r_i)$ denotes the hierarchy set that role r_i is in. Then Eq. (6) should be rewritten as Eq. (25).

$$G(T_{rti}) = [gt, gt + t \cdot e] \sqsubseteq r \cdot D\&\& \\ r \cdot rid \in \{r_j \cdot rid | r_j = \min(H_i),\ 1 \le i \le Q\}. \tag{25}$$

(ii) *Disregarding duty constraints.*

The SoD constraint is captured by adding the P_{seq} place (for sequential tasks) or the P_{par} place (for parallel tasks) as well as the corresponding arcs. When the duty constraints are disregarded, the corresponding place and arcs are deleted. The corresponding guard functions shown in Eqs. (8)–(11) are also removed.

(iii) *Disregarding role hierarchy constraints.*

When disregarding the role hierarchy constraint for task t_i, Eq. (6) should be changed to Eq. (26). In the equation, the term for enforcing the role hierarchy constraint is removed. However, the expression for enforcing the role constraint is omitted in Eq. (6) only because the role hierarchy constraint is treated as a special case of the role constraint. Since the term for the role hierarchy constraint is now removed, the term for the role constraint should be added.

$$G(T_{rti}) = r \cdot rid \in R(ti)\&\&[gt, gt + t \cdot e] \sqsubseteq r \cdot D. \tag{26}$$

5.2. Calculating other performance metrics

The CPN Tools can evaluate the overall performance of a constructed Petri-net model in terms of various metrics. For example, resource utilization can be evaluated by monitoring the number of tokens in the place corresponding to the resource pool (which is P_{rp} or P_u in the model). Assume the initial marking of the place P_{rp} is n, and during model simulations the average number of tokens in the place observed by the CPN Tools is avg_n, then the Utilization of Computing Resources (*UCR*), can be calculated as:

$$UCR = 1 - avg\_n/n.$$

The Utilization of Human Resources, denoted as *UHR*, can be calculated in the similar way.

It is also straightforward to determine the response time of a workflow using the CPN Tools. The tool can extract the time stamp when a workflow arrives at the system, and the time stamp when the last task in the workflow is completed. The difference between these two time stamps is the response time of the workflow. Based on the information, we can easily calculate the mean response time of all workflows. Other performance metrics that the CPN Tools is able to evaluate include throughput, deadline miss rate, etc.

5.3. Improving performance under the authorization control

Generally speaking, there are two possible approaches to improving performance in the presence of authorization policies. On the one hand, the company can relax the authorization constraints which are causing the significant overhead. For example, the company can allow the users to share an additional common role and allow this role to be always activated. On the other hand, the company can change the system deployment which is not directly related to the authorization control, such as the scheduling strategy, and the quantity of resources.

The first approach is not always feasible since the presence of the authorization constraints may be mandatory for security reasons. However, if some authorization constraints are allowed to be relaxed in certain circumstances, we can apply the methods presented in Section 5.1 to calculate the authorization overhead caused by each authorization constraint, and therefore determine which authorization constraints are significantly hampering the performance. Which authorization constraints can be relaxed mainly depends on the nature of the businesses being processed and the company policy, which are out of the scope of this paper.

In this paper, we focus on the second approach, i.e., improving the performance by (1) adjusting the scheduling strategy, and (2) changing the quantity of human and computing resources.

5.3.1. Improving performance by changing the scheduling strategy

The scheduling strategy can have impact on the cardinality overhead. When the number of the tasks running under a role has reached the number set by the cardinality constraint, the new task requiring that role has to wait until one of those tasks is completed. The waiting time is regarded as the cardinality overhead. A good scheduling strategy could reduce the tasks' waiting time caused by the cardinality constraints and therefore improve performance. The First-Come-First-Served scheduling strategy is employed in the model presented in Section 4. In the rest of this Subsection, a new scheduling strategy is proposed to enhance the performance by reducing the cardinality overhead.

Assume that r_i's cardinality constraint is c_i. In order to eliminate r_i's cardinality overhead, the number of the tasks running under role r_i should be less than c_i when a new task arrives requesting role r_i. γ_i denotes the arrival rate of the tasks assigned to role r_i, and RT_i denotes the mean response time of the tasks assuming r_i. γ_i can be obtained from the simulation of the authorization model. RT_i can be determined as follows. According to Little's Law in the Queuing theory [37], we have $c_i = \gamma_i \times RT_i$. Therefore, in order to eliminate r_i's cardinality overhead, RT_i should be less than c_i/γ_i, which is regarded as the desired average response time of the tasks assigned to r_i.

The proposed new scheduling strategy is essentially an EDF (Earliest Deadline First) scheduler. Assume N types of workflow are modeled in the system. WF_i denotes workflow i, t_{ij} denotes task t_j in WF_i and e_{ij} denotes t_{ij}'s execution time. The instances of WF_i are initiated following the Poisson process with the initiation rate of λ_i. During the model simulations, we obtain the following statistical information.

1: Obtaining which tasks assume role r_i during the model simulation.

2: Calculating the average execution time of the tasks running under r_i.

3: Calculating the ratio of the desired average response time (i.e., c_i/γ_i) to the average execution time of the tasks assigned to r_i. The ratio is denoted as η_i. We assume that the system is in a steady state. So η_i should be greater than 1.

In a new model simulation, when task t_{jk} in WF_k is assigned to role r_i, a soft deadline is attached to t_{jk}, denoted as d_{jk}. d_{jk} is set as

$$d_{jk} = rt_{jk}@ts + e_{ij} \times \eta_i, \tag{27}$$

where $rt_{jk}@ts$ is the timestamp of the token rt for task t_{jk}, which is the time when the token rt is deposited into the place P_{rt}.

In the proposed EDF scheduling, the task with the smallest value of $d_{ij} - gt$ is always first put into execution (gt is the current global time of the model). By doing so, the tasks which are more likely to cause the cardinality overhead will be put into execution first. The effectiveness of the EDF scheduling is demonstrated in Section 6.

If the EDF scheduling is applied, the term for enforcing the scheduling strategy in $G(T_{cex})$ (defined in Eq. (19)) and $G(T_{hex})$ (in Eq. (21)) should be rewritten as Eq. (28), where $\tau.d$ is the deadline of the token τ (specified in Eq. (27)).

$$\tau \in \{\tau \mid \tau \cdot d = \min\{\tau \cdot d\}\}. \tag{28}$$

5.3.2. Improving performance by adjusting resources

The time that a task waits for computing resources can be calculated as the duration between the time when a token is deposited into the P_{urt} place and the time when the T_{cex} transition fires. From the model simulations, we can calculate the average resource waiting time of computing tasks (including HCTs and ACTs), which is denoted as w^{CT}.

The time for an HT to wait for human resources can be calculated as the duration between the time when a role is assigned to the HT and the time when a user starts processing the HT. During the model simulations, we can calculate the average resource waiting time of the HTs assuming the same role, r_i, which is denoted as w_i^{HT}, as well as the average resource waiting time of all HTs, denoted as w^{HT}.

The tasks' resource waiting time can be regarded as the overhead caused by the resource contention. The HTs can only be processed by the users associated with eligible roles. The roles with high resource waiting times are the *bottleneck* roles that present higher overhead, and therefore may jeopardize the performance. Given two roles, r_i and r_j ($i \neq j$), if w_i^{HT} is greater than w_j^{HT}, it indicates that the human resources provisioned for role r_i are not as adequate as those for r_j under the current authorization settings. Similarly, by comparing w^{HT} and w^{CT}, we can know whether the system provides the balanced human resources and computing resources, given the current authorization constraints.

When planning the resource capacity for a system, it is desired to achieve the balanced performance in terms of w_i^{HT} ($1 \leq i \leq R$, R is the number of roles) and also in terms of w^{HT} and w^{CT}. The following method is proposed to approximate the amount of human and computing resources so that the balanced resource waiting time can be achieved.

From the model simulations, we can gather the arrival rate of the HTs assuming role r_i, denoted as λ_i^{HT}, and the average execution time of the HTs that assume r_i, denoted as e_i. Assume that we aim to configure the system capacity so that w_i^{HT} ($1 \leq i \leq R$) and w^{CT} have the values close to w. According to the Queuing theory [37], w satisfies Eq. (29), where n_i is the number of homogeneous human resources (i.e., the users) associated with role r_i. Eq. (29) can be transformed to Eq. (30) to calculate n_i.

$$w = \frac{\lambda_i^{HT} e_i^2}{n_i(n_i - e_i\lambda_i^{HT})} \tag{29}$$

$$n_i = \sqrt{\lambda_i^{HT} e_i^2 \left(\frac{\lambda_i^{HT}}{4} + \frac{1}{w}\right)} + \frac{\lambda_i^{HT} e_i}{2}. \tag{30}$$

This paper assumes that a user may be associated with multiple roles. Out of all HTs assigned to u_i, χ_{ij} denotes the proportion of the tasks that assume r_i. Again, χ_{ij} can be obtained from the model simulations. The number of human resources provided for r_i is regarded as $\sum_{j=1}^{|U(r_i)|} \chi_{ij}$, where $|U(r_i)|$ is the number of users associated with r_i; χ_{ij} is 1 if u_j is only associated with r_i. In order to reduce the resource waiting time from w_i^{HT} to w, the additional number of human resources that should be provided can be calculated as $n_i - \sum_{j=1}^{U_i} \chi_{ij}$. If the result is not an integer, it is rounded to the closest integer.

Similarly, we can have the following equation for the computing tasks, where λ^{CT} is the rate at which the computing tasks arrive for computing resources, e_{ct} is the average execution time of computing tasks, n_{ct} is the number of computing resources.

$$w = \frac{\lambda^{CT} e_{ct}^2}{n_{ct}(n_{ct} - e_{ct}\lambda^{CT})}. \tag{31}$$

With Eq. (31), we can calculate n_{ct} such that the average resource waiting time of the computing tasks is w. Assume N is the current number of computing resources. Then $n_{ct} - N$ is the additional number of computing resources that should be equipped in order to achieve the desired performance of w.

After calculating the new resource parameters, we can then adjust the resource quantity in the models and re-run the model simulation to verify the performance achieved by the new system capacity under the authorization settings.

Although we consider homogeneous human resources in this paper, it is easy to extend the calculations to the heterogeneous case.

6. Experimental studies

This section presents the simulation experiments to demonstrate the impact of the authorization constraints on the performance in terms of mean Response Time (RT) of workflows, Utilization of Computing Resources (UCR) and Utilization of Human Resources (UHR). The performance in terms of deadline miss rate and throughput is correlated with response time and utilization, respectively.

In the simulations presented in this paper, the workflows are randomly generated, each workflow containing *TNUM* tasks and each task in a workflow having the maximum of *MAX_DG* children. A workflow contains three types of task, HT, HCT and ACT, following a certain ratio of the number of tasks in each type (denoted as |HT| : |HCT| : |ACT|). *RNUM* roles and *UNUM* users are assumed to be involved in processing the workflows.

The role constraints (i.e., the set of roles that a task can assume) for each HT and HCT are set in the following fashion. The simulation sets a maximum number of roles that any task can assume in the role constraints, denoted as *MAX_RCST*, which represents the level of restrictions imposed on the role assignment for tasks. When setting the role constraint for task t_i, the number of roles that can run t_i is randomly selected from [1, *MAX_RCST*], and then that number of roles are randomly selected from the role set. A similar scheme is used to associate users to roles. The maximum number of users a role can be associated to is denoted as *MAX_U2R*. The number of users belonging to role r_i is randomly selected from [1, *MAX_U2R*]; and these users are then randomly selected for r_i from the user set.

NUM_SoD and *NUM_BoD* denote the number of tasks that are associated with *SoD* and *BoD* constraints, respectively. Duty constraints were set as follows. Each time, two tasks are randomly selected from the workflow to establish the BoD constraint between them until *NUM_BoD* tasks are covered. And then the same procedure is applied to establish the SoD constraints among tasks. In this process, the method presented in [29] is used to guarantee that the designated duty constraints on these selected tasks can be satisfied. We assume that the arrival of workflow instances follows the Poisson process and that the tasks' execution times follow an exponential distribution. The human tasks have the average execution time of *EX_H* time units, while the computing tasks, including HCT and ACT, have the average execution time of *EX_C* units. There are *NR* homogeneous computing resources.

CARD denotes the cardinality constraint, i.e., the maximum number of the tasks that can be run simultaneously in the system by a role. The temporal constraints on roles are set in the following way. For each role, a time duration is selected from a period of *TD* time units. The selected time duration occupies the specified percentage of the *TD* time units, which is denoted as *TEMP*. The starting time of the selected duration is chosen randomly from the range of $[0, TD \times (1 - TEMP)]$. For example, if $TD = 200$ and $TEMP = 70\%$, the starting point is randomly selected from 00% to $30\% \times 200$.

The role hierarchy is set as follows. The *RNUM* roles are randomly divided into *H* hierarchy sets. The number of roles in each of the first $(RNUM - 1)$ sets is $\lfloor RNUM/H \rfloor$, and in the last set the number of roles is $RNUM - \lfloor RNUM/H \rfloor \times (RNUM - 1)$. Then in each set, the roles are randomly selected. The sequence of the roles being selected is used to determine the role hierarchy. The role being selected first has the lowest privilege.

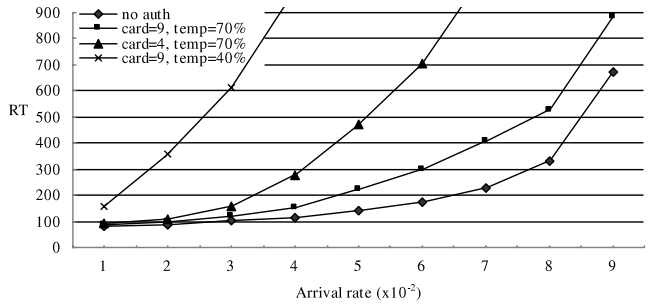
6.1. Impact of cardinality and temporal constraints

Fig. 8 shows the RT and the RU of running the workflows under authorization constraints as the arrival rate of the workflows increases. Fig. 8 also shows the impact caused by changing the cardinality and the temporal constraints. Fig. 8(a) shows the performance in terms of RT, while Fig. 8(b) and (c) show the performance in terms of UCR and UHR, respectively. In order to obtain the baseline performance for comparison, we simulated the workflow scheduling and execution process assuming there is no any authorization policy, that is, the ready HTs (or HCTs and ACTs) are scheduled to run on any of the idle users (or computing resources) on a FCFS basis. The recorded performance data is also presented in Fig. 8 (the "no auth" curve).

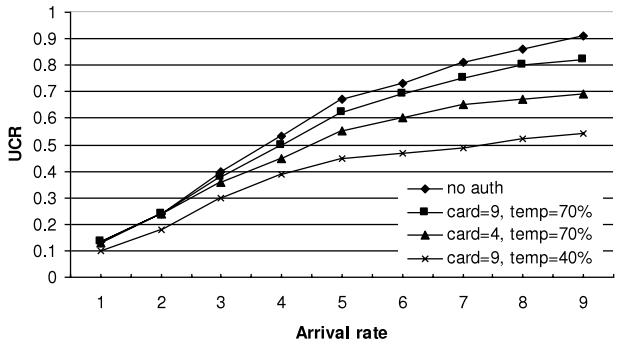
By comparing the "no auth" curve and the "card = 9, temp = 70%" curve in Fig. 8(a), it can be seen that when the authorization constraints are imposed, RT increases. This is because when applying the authorization constraints, the tasks have to wait not only for resources, but also for the assignment of roles, which may not be available due to temporal constraints or cardinality constraints.

When the cardinality constraint varies from 9 to 4, the performance becomes even worse, which is to be expected. A closer observation shows that when the arrival rate of the workflows is low (no more than 3×10^{-2}), the change in the cardinality constraint does not have obvious impact on the RT, while the RT increases dramatically as the arrival rate becomes higher. This is because when the arrival rate is low, the number of tasks trying to run simultaneously in the system is small. Therefore, reducing the cardinality number from 9 to 4 may not affect the performance. However, when the arrival rate is high, the small cardinality number may form a performance bottleneck in the system, and the incoming tasks may have to wait a long time for role assignment since all role instances are assigned to the existing tasks in the system.

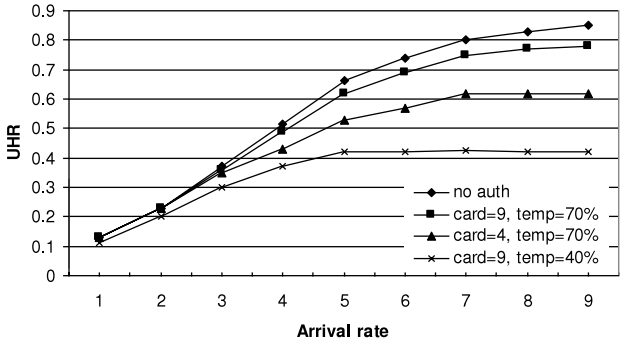
It can also be observed from Fig. 8(a) that compared with the impact of changing the cardinality constraints, the impact of changing the roles' temporal availability is different. When the temporal constraint varies from 70% to 40%, the RT has noticeable increases even when the arrival rate is low. This may be because



(a) Impact on RT.



(b) Impact on UCR.



(c) Impact on UHR.

Fig. 8. The impact of Authorization constraints on (a) mean response time of the workflows, (2) utilization of computing resources, and (3) utilization of human resources; "no auth" means there are no authorization constraints; "card = 9" and "card = 4" mean that the maximum number of tasks that a role can run simultaneously is 9 and 4, respectively; $TNUM = 16$, $|HT|:|HCT|:|ACT| = 7:7:2$, $MAX\_DG = 4$, $EX\_C = 18$, $EX\_H = 18$, $RNUM = 6$, $UNUM = 18$, $MAX\_U2R = 6$, $MAX\_RCST = 4$, $NUM\_SoD = 3$, $NUM\_BoD = 3$, $TD = 200$, $H = 2$, $NR = 16$.

when the roles' availability periods are shorter, it is more likely that the required roles are not available when a task requests the role assignment, no matter how low the tasks' arrival rate is. This result suggests that the temporal constraints have bigger impact on the RT than the cardinality constraints, especially when the workload level is low.

Through the modeling approach and the simulation results as in Fig. 8, we are able to acquire the quantitative insight into how much impact the authorization constraints can have. For example, suppose the workflows' RT is desired to be no more than 200 time units. When there are not authorization constraints, the system can accommodate a workflow stream with a mean arrival rate of up to 0.065. However, when there are authorization constraints, the workload level that the system is able to handle is reduced to approximately 0.047, 0.033, and 0.012 in the case of "card = 9, temp = 70%", "card = 4, temp = 70%" and "card = 9, temp = 40%", respectively.

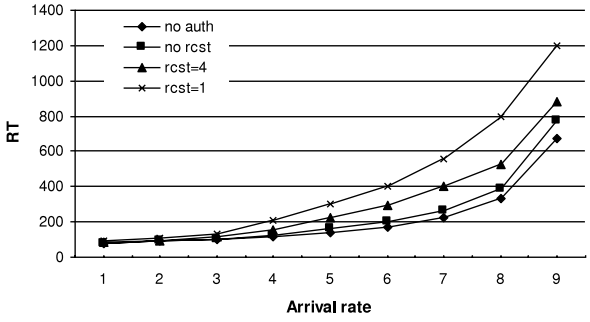
It can be observed from Fig. 8(b) and (c) that the utilization of both computing resources and human resources decreases when the authorization constraints are present, and that when the constraints are tighter, the utilization decreases by a larger extent. This is because when there are authorization constraints, the tasks may have to wait for role assignment due to roles' unavailability even if there are free resources. Consequently, the utilization cannot be as high as when there are not authorization constraints. Further, when the constraint is tighter, it is more likely that such a situation will occur.

By comparing Fig. 8(b) and (c), it can be seen that the curve trends are different in these two figures. In Fig. 8(c), UHR becomes almost constant when the arrival rate reaches a certain level, while UCR keeps increasing in Fig. 8(b) as the arrival rate increases from 0.01 to 0.09. This can be explained as follows. In Fig. 8(c), when the arrival rate of the workflows increases, the arrival rate of HTs increases. When the arrival rate reaches a certain level, the authorization constraints will become a bottleneck for HTs' executions, i.e., the time that HTs wait for human resources is negligible while they spend a long time in waiting for role assignment. In this situation, the utilization will not increase no matter how high the workflows' arrival rate is. In Fig. 8(b), when the workflows' arrival rate increases to a point where the authorization constraints form a bottleneck, the UCR consumed by HCTs will become constant. However, the arrival rate of ACTs keeps increasing, and ACTs do not require authorization. Further, since the utilization consumed by HCTs is capped, there are free resources left to run ACTs. Therefore, UCR will continue to increase as the workflows' arrival rate (also ACTs' arrival rate) increases.

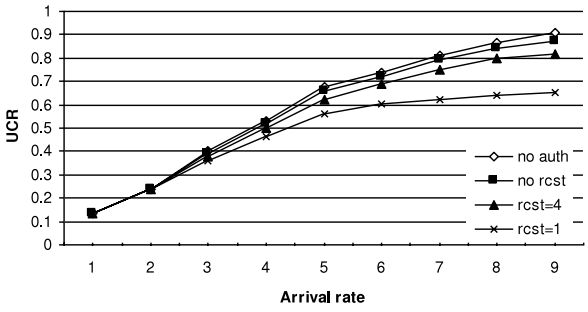
6.2. Impact of role constraints

Fig. 9 shows the impact of role constraints in terms of RT, where "no rcst" means that there is no role constraint applied to the authorization, but other types of constraint are still in present and the settings are the same as in Fig. 8. As can be seen from this figure, when there are no role constraints (note that there are still other constraints), the RT is very close to that achieved when there are no authorization constraints at all. This result can be explained as follows. Since there are not the role constraints, the tasks can assume any role (only subject to the duty and role hierarchy constraints). Consequently, when one role is not available (either due to temporal constraint or cardinality constraint) for running a task, it is very likely that another role is available.

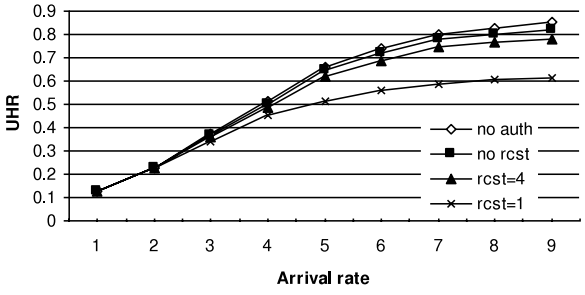
The simulation is also conducted at the other extreme, i.e., setting *MAX_RCST* to be 1, which means that a task can only assume one particular role. The performance in terms of RT corresponds to



(a) Impact on RT.



(b) Impact on UCR.



(c) Impact on UHR.

Fig. 9. Impact of role constraints in terms of RT; the simulation setting is the same as those in Fig. 8.

the "rcst = 1" curve in Fig. 9. It can be seen that the RT is much worse than other cases. This is due to the fact that every task is restricted to run under a particular role. If that role is not available, the task has no choice but to wait for authorization.

Fig. 9(b) and (c) show the impact of the role constraints on UCR and UHR, respectively. It can be observed that when the role constraints become stricter, both UCR and UHR decrease. Once again, this is because the tasks have to wait for roles' availability, which causes the otherwise busy resources to be idle.

6.3. Impact of duty constraints

Fig. 10(a) and (b) show the impact of the BoD and SoD constraints, respectively, on the RT. The results on UCR and UHR are omitted, since the impact of duty constraints on UCR and UHR has the similar trend as those presented in previous figures. Namely, if the constraint has big impact on RT (causing RT to increase), then it also has big impact on utilization (causing utilization to decrease).

It can be seen from Fig. 10(a) that when *NUM_BoD* is 8, the RT deteriorates dramatically, especially when the arrival rate is high. This result can be explained as follows. When *NUM_BoD* is 8, at least 8 tasks in a workflow will be run under the same role. Therefore, the workload is not balanced across different roles, which harms the performance. Moreover, if that role is not activated due to its temporal constraint, the tasks have to wait for its activation. This result suggests that BoD has big impact on the workflow performance.

Fig. 10(b) shows that the SoD constraints have a different impact level. When the *NUM_SoD* increases from 3 to 8, the RT has little increase. This is because the SoD constraint only rules out one role for authorization and the rest of the roles are still eligible.

6.4. The authorization overhead caused by different types of constraints

Fig. 11 shows the authorization overhead caused by different types of constraint. As can be observed from this figure, the results in this figure are consistent with those presented in previous figures. For example, SoD causes little overhead.

It can also be seen from this figure that the role hierarchy constraint causes the overhead too. This is because when the role hierarchy constraint is applied, more tasks assume the roles with lower privileges, which can result in higher resource waiting time for those roles, and may even cause the tasks to wait for role assignment due to the cardinality constraints.

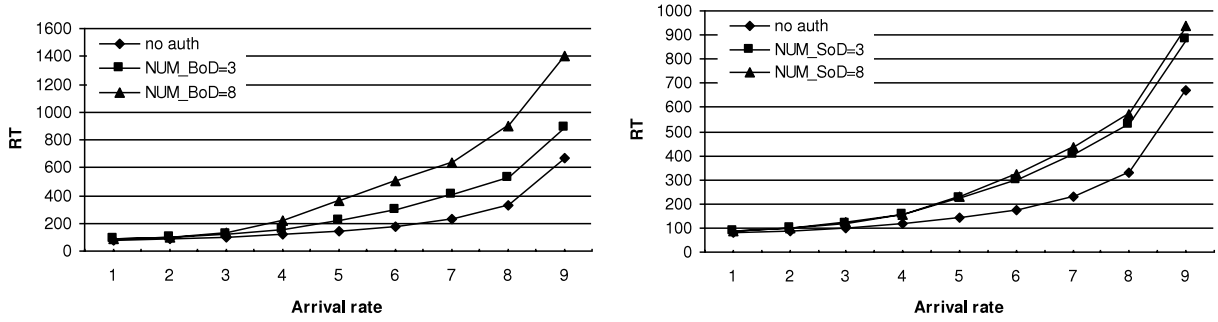
6.5. Impact of the EDF scheduling strategy

Fig. 12 compares the RT achieved by the FCFS scheduling and the proposed EDF scheduling.

It can be seen from this figure that the EDF scheduling outperforms the FCFS scheduling when the arrival rate is greater than 0.04. The results can be explained as follows. When the arrival rate is low, the number of tasks running in the system is low. Therefore, the cardinality constraint will not pose as a bottleneck and the EDF scheduling cannot manifest its effectiveness. When the arrival rate is bigger, the proposed EDF scheduling takes into account the cardinality number and the tasks' arrival rate for roles, and tends to first process the tasks assigned to the role with the smaller cardinality number. Therefore, it reduces the chance that the number of tasks running under a role reaches its specified cardinality number. A closer observation from this figure shows that when the arrival rate further increases (more than 0.08), the advantage of the EDF scheduling diminishes. This may be because when the arrival rate is very high, the number of the tasks running under most roles reaches their cardinality numbers and under this circumstance, the contribution of the EDF scheduling is limited.

6.6. Impact of changing resources

The impact of changing resources is investigated using the following method. We first conduct the model simulation to obtain the values of w_i^{HT} $(1 \leq i \leq RNUM)$, w^{HT} and w^{CT} (which are discussed in Section 5.3.2), given the current resource parameters. Then we set the expected waiting time w and use the



(a) Impact of BoD. (b) Impact of SoD.

Fig. 10. The impact of duty constraints; the simulation settings are the same as in Fig. 8.

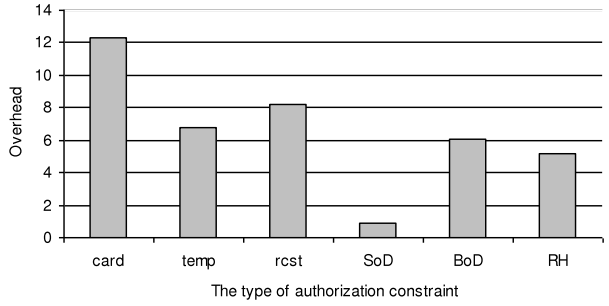


Fig. 11. The authorization overhead caused by different types of authorization constraints; the workflow arrival rate is 0.05; CARD = 9; TEMP = 70%; other settings are the same as in Fig. 8.

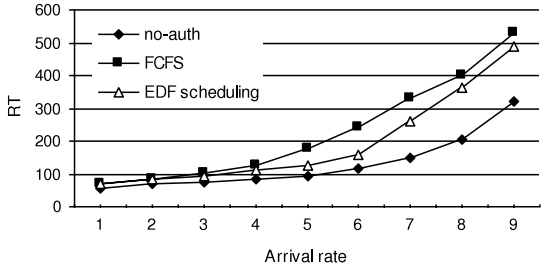


Fig. 12. The impact of the EDF scheduling.

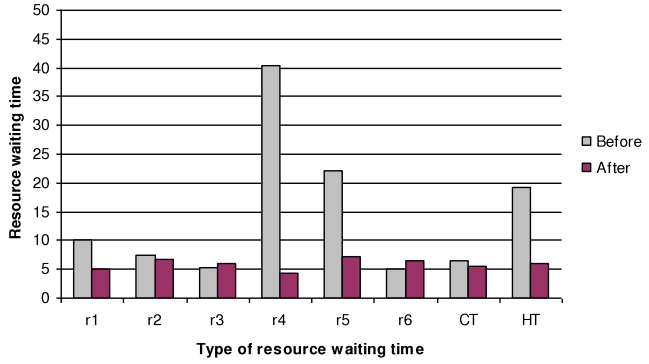


Fig. 13. Resource waiting time; the value at r_i $(1 \leq i \leq 6)$ corresponds to w_i^{HT}, the value at CT is w^{CT}), and the value at HT is w^{HT}; the arrival rate of the workflows is 0.08; each role is set to have 3 users; the execution time of each task is 18; other experimental settings are the same as in Fig. 8.

methods presented in Section 5.3.2 to change the resource quantity accordingly. Finally, we conduct the model simulations again to verify that the new resource parameters can achieve the desired performance.

Table 3
The number of resources before and after the adjustment.

	r_1	r_2	r_3	r_4	r_5	R_6	RP
Before	3	3	3	3	3	3	16
After	4	3	3	5	4	2	16

Fig. 13 shows the values of w_i^{HT}, w^{CT}, and w^{HT} before and after adjusting resources. The value at r_i $(1 \leq i \leq 6)$ in the figure corresponds to the average resource waiting time of the HTs assuming r_i (i.e., w_i^{HT}), the value at CT is the average resource waiting time of computing tasks (i.e., w^{CT}), and the value at HT is the average resource waiting time of all HTs (i.e., w^{HT}). In these simulations, in order to investigate the impact of authorization constraints on the tasks' resource waiting time, each role is set to have the same number of users (set to be 3) and each task has the same execution time (set to be 18).

As can be seen from the "Before" data in this figure, although each role has the same number of human resources and the tasks have the same execution time, different roles have different average resource waiting times. This is because the authorization constraints may cause the unbalanced task arrival rates among roles. This result suggests again that the authorization constraints have impact on workflow performance. Further, it can be seen from the "Before" data in the figure that w^{CT} is less than w^{HT}. This indicates that under the current authorization constraints, the capacity of the human resources is not adequate compared with the computing resources, and therefore forms the bottleneck in the system.

After we obtain the performance of the current resource capacity in the system, we set the desired resource waiting time, w, to be the average resource waiting time of the computing tasks, i.e., $w = w^{CT}$. We then apply the methods proposed in Section 5.3.2 to adjust the number of human resources for each role, so that $w_i^{HT} = w$. Table 3 shows the number of resources before and after the adjustment, where RP is the number of computing resources. The columns labeled as "After" in Fig. 13 show the corresponding resource waiting times after the resource adjustment.

As can be seen from the "After" data in Fig. 13, w_i^{HT} and w^{CT} have the similar values after adjusting the resources as shown in Table 3. It shows that changing resources can significantly affect the performance, and that the resource adjustment method proposed in this paper is able to achieve balanced performance across roles as well as between human resources and computing resources in the system.

7. Conclusions

This paper models the authorization and execution of hybrid workflows consisting of human and computing tasks. The impact of authorization on workflow executions is analyzed. The Timed

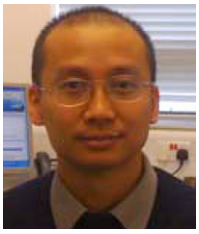
Color Petri Nets (TCPN) formalism is employed to construct the models. Various authorization constraints are modeled in this paper, including role, temporal, cardinality, separation of duty, binding of duty, and role hierarchy constraints. The model is constructed in a modular fashion so that the model construction can be automated; this makes it easy to model a large collection of authorization policies and complex workflows. The modeling mechanism has been implemented using the CPN Tools. Authorization overhead and various performance metrics can be computed from the constructed model, including those for system-oriented performance and application-oriented performance. This paper also proposes the methods to improve performance under the authorization constraints.

Acknowledgments

This work is supported by the Leverhulme Trust (grant number RPG-101), the National High-tech R&D Program of China (863 Program, Grant No. 2011AA01A203), the Key Program of National Natural Science Foundation of China (61133005), the National Natural Science Foundation of China (grant numbers: 61173166 and 60803130).

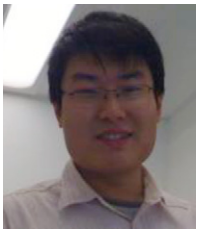
References

[1] E. Deelman, D. Gannon, M. Shields, I. Taylor, Workflows and e-science: an overview of workflow system features and capabilities, Future Generation Computer Systems 25 (5) (2009) 528–540.
[2] L. He, S.A. Jarvis, D.P. Spooner, H. Jiang, D.N. Dillenberger, G. Nudd, Allocating non-real-time and soft real-time jobs in multiclusters, IEEE Transactions on Parallel and Distributed Systems 17 (2) (2006) 99–112.
[3] C. Hsu, K. Huang, F. Wang, Online scheduling of workflow applications in grid environments, Future Generation Computer Systems 27 (6) (2011) 860–870.
[4] Web services business process execution language version 2.0. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf, 2007.
[5] K. Gaaloul, A. Schaad, U. Flegel, F. Charoy, A secure task delegation model for workflows, in: The Second International Conference on Emerging Security Information, Systems and Technologies, 2008, pp. 10–15.
[6] T. Hara, T. Arai, Y. Shimomura, T. Sakao, Service CAD system to integrate product and human activity for total value, CIRP Journal of Manufacturing Science and Technology 1 (4) (2009) 262–271.
[7] D. Schall, S. Dustdar, M. Blake, Programming human and software-based web services, IEEE Computer 43 (7) (2010) 82–85.
[8] Q. Zhao, X. Liu, D. Sun, T. Liu, Y. Li, Mashing-up rich user interfaces for human interation in WS-BPEL, in: The 2010 IEEE International Conference on Web Services, 2010, pp. 559–566.
[9] Video management workflow. http://www.telestream.net/pdfs/whitepapers/wp-video-workflow-management.pdf, 2010.
[10] Web services — human task (WS-HumanTask) specification version 1.1. http://docs.oasis-open.org/bpel4people/ws-humantask-1.1.html, 2010.
[11] WS-BPEL extension for people (BPEL4People) specification version 1.1. http://docs.oasis-open.org/bpel4people/bpel4people-1.1-spec-cd-06.pdf, 2009.
[12] G. Ahn, R. Sandhu, Role-based authorization constraints specification, ACM Transactions on Information and System Security 3 (4) (2000).
[13] E. Bertino, J. Crampton, F. Paci, Access control and authorization constraints for ws-bpel, International Conference on Web Services (2006) 275–284.
[14] J. Crampton, M. Huth, On the modeling and verification of security-aware and process-aware information systems, Business Process Management Workshops 100 (6) (2012) 423–434.
[15] J. Joshi, E. Bertino, U. Latif, A. Ghafoor, A generalized temporal role-based access control model, IEEE Transactions on Knowledge and Data Engineering 17 (1) (2005) 4–23.
[16] Y. Lu, L. Zhang, J. Sun, Using colored petri nets to model and analyze workflow with separation of duty constraints, International Journal of Advanced Manufacturing Technology 40 (1–2) (2009) 179–192.
[17] X. Zhao, Z. Qiu, C. Cai, H. Yang, A formal model of human workflow, in: The 2008 IEEE Intl. Conf. on Web Services, pp. 195–202.
[18] D. Zou, L. He, H. Jin, X. Chen, CRBAC: imposing multi-grained constraints on the rbac model in the multi-application environment, Journal of Network and Computer Applications 32 (2) (2009) 402–411.
[19] M. Stuit, H. Wortmann, N. Szirbik, J. Roodenburg, Multi-view interaction modelling of human collaboration processes: a business process study of head and neck cancer care in a dutch academic hospital, Journal of Biomedical Informatics 44 (6) (2011) 1039–1055.
[20] Y. Jin, S. Reveliotis, A generalized stochastic petri net model for performance analysis and control of capacitated reentrant lines, IEEE Transactions on Robotics and Automation 19 (3) (2003) 474–480.
[21] K. Jensen, L. Kristensen, Coloured Petri Nets. Modeling and Validation of Concurrent Systems, Springer-Verlag, 2009.
[22] K. Jensen, L. Kristensen, L. Wells, Coloured petri nets and cpn tools for modeling and validation of concurrent systems, International Journal on Software Tools for Technology Transfer 9 (3) (2007) 213–254.
[23] D. Chakraborty, V. Mankar, A. Nanavati, Enabling runtime adaptation of workflows to external events in enterprise environments, in: The 2007 IEEE International Conference on Web Services, pp. 1112–1119.
[24] P. Delias, A. Doulamis, N. Doulamis, N. Matsatsinis, Optimizing resource conflicts in workflow management systems, IEEE Transactions on Knowledge and Data Engineering 23 (3) (2011) 417–432.
[25] V. Atluri, W. Huang, A Petri net based safety analysis of workflow authorization models, Journal of Computer Security 8 (2/3) (2000) 209–240.
[26] L. He, K. Duan, X. Chen, D. Zou, Z. Han, A. Fadavinia, S. Jarvis, Modelling workflow executions under role-based authorization control, in: IEEE International Conference on Service Computing, SCC 2011, 2011.
[27] S. Manolache, Schedulability Analysis of Real-Time Systems with Stochastic Task Execution Times, Ph.D. Thesis, Department of Computer and Information Science, IDA, Linkoping University, 2002.
[28] Q. Wang, N. Li, Satisfiability and resiliency in workflow authorization systems, ACM Transactions on Information and System Security 13 (4) (2010) 1–35.
[29] J. Crampton, A reference monitor for workflow systems with constrained task execution, in: Proceedings of the Tenth ACM Symposium on Access Control Models and Technologies, 2005, pp. 38–47.
[30] J. Wainer, P. Barthelmess, A. Kumar, W-RBAC — a workflow security model incorporating controlled overriding of constraints, International Journal of Cooperative Information Systems 12 (4) (2003) 455–486.
[31] W. Zuberek, Timed petri nets in modeling and analysis of cluster tools, IEEE Transactions on Robotics and Automation 17 (2001) 562–575.
[32] V. Atluri, J. Warner, Supporting conditional delegation in secure workflow management systems, in: Proceedings of the Tenth ACM Symposium on Access Control Models and Technologies, 2005.
[33] P. Hung, K. Karlapalem, A secure workflow model, in: The Australasian Information Security Workshop, AISW 2003, 2003.
[34] L. He, M. Calleja, M. Hayes, S.A. Jarvis, Performance prediction for running workflows under role-based authorization mechanisms, in: Proc. of the 2009 IEEE International Symposium on Parallel & Distributed Processing, IEEE Computer Society Press, 2009, pp. 1–8.
[35] M. Owen, J. Raj, BPMN and business process management, http://www.bpmn.org/Documents/6AD5D16960.BPMN_and_BPM.pdf.
[36] http://msdn.microsoft.com/en-us/library/bb330937.asp.
[37] L. Kleinrock, Queueing System, John Wiley & Sons, 1975.



Ligang He is an Associate Professor in the Department of Computer Science at the University of Warwick. He studied for the Ph.D. degree in Computer Science at the University of Warwick, UK, from 2002 to 2005, and then worked as a post-doctor in the University of Cambridge, UK. In 2006, he joined the Department of Computer Science at the University of Warwick as an Assistant Professor. His research interests focus on parallel and distributed processing, Cluster, Grid and Cloud computing. He has published more than 40 papers in international conferences and journals, such as IEEE Transactions on Parallel and Distributed Systems, IPDPS, Cluster, CCGrid, MASCOTS. He has been a member of the program committee for many international conferences, and has been the reviewer for a number of international journals, including IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Computers, IEEE Transactions on Automation Science and Engineering, etc. He is a member of the IEEE.

Chenlin Huang is an Associate Professor in the School of Computer Science at the National University of Defense Technology, China. His research areas are operating systems, security and high performance computing.



Kewei Duan is currently a Ph.D. student in the Department of Computer Science at the University of Bath, UK. His research interests are in performance modeling and evaluations and service computing.

Kenli Li is a Professor in the School of Computer and Communication at the Hunan University, China. He received his Ph.D. Degree in Computer Science from Huazhong University of Science and Technology, China in 2003. His research interests are parallel and distributed computing, real-time and embedded systems.

Hao Chen is an Associate Professor in the School of Computer and Communication at the Hunan University, China. He received the Ph.D. Degree in Computer Science from Huazhong University of Science and Technology, China in 2005. His research interests include virtual machines, operating systems, distributed and parallel computing and security. He is a member of the IEEE.

Jianhua Sun is an Associate Professor in the School of Computer and Communication at the Hunan University, China. She received the Ph.D. Degree in Computer Science from Huazhong University of Science and Technology, China in 2005. Her research interests are in security and operating systems.

Stephen A. Jarvis is Professor of High Performance and Distributed Computing at the University of Warwick and is co-organiser for one of the UK's High End Scientific Computing Training Centres. He has authored more than 130 refereed publications (including three books) and has been a member of more than 50 programme committees for IEEE/ACM international conferences and workshops since 2003, including: IPDPS, HPDC, CCGrid, SC, MASCOTS, DSN, ICPP. He is a former member of the University of Oxford Computing Laboratory, and in 2009 was awarded a prestigious Royal Society Industry Fellowship in support of his industry-focused work on high-performance computing.