**CERIAS Tech Report 2006-14**

# POLICY-BASED VERIFICATION OF DISTRIBUTED WORKFLOWS IN A MULTI-DOMAIN ENVIRONMENT

by Basit Shafiq, Ammar Masood, and Arif Ghafoor

Center for Education and Research in
Information Assurance and Security,
Purdue University, West Lafayette, IN 47907-2086

# Policy-Based Verification of Distributed Workflows in a Multi-Domain Environment

Basit Shafiq, Ammar Masood, Arif Ghafoor
Purdue University, West Lafayette IN 47907, USA

## Abstract

There is a growing need to support secure interaction among autonomous domains/systems for developing distributed applications. As domains operate according to their individual security and access control policies, supporting secure interactions among domains for distributed workflows is a complex task prone to subtle errors that can have serious security implications. In this paper we propose a framework for verifying secure composibility of distributed workflows in an autonomous multi-domain environment. The objective of workflow composibility verification is to ensure that all the users or processes executing the designated workflow tasks conform to the security policy specifications of all collaborating domains. A key aspect of such verification is to determine the time-dependent schedulability of distributed workflows, assumed to be invoked on a recurrent basis. We use a two-step approach for verifying secure workflow composibility. In the first step, a distributed workflow is decomposed into domain-specific projected workflows and is verified for conformance with the respective domain's security and access control policy. In the second step, the cross-domain dependencies amongst the workflow tasks performed by different collaborating domains are verified.

---

Portions of this work were supported by Grant IIS-0209111 from the National Science Foundation and by sponsors of the Center for Education and Research in Information Assurance and Security.

# 1 Introduction

The rapid proliferation of the Internet and cost-effective growth of its key enabling technologies have created unprecedented opportunities for developing large-scale collaborative workflow-based applications. These applications require access to information and computational resources owned by autonomous and heterogeneous domains and service providers. Supporting distributed workflows in such environments is a challenging task. The individual domains are autonomous in the sense that they operate according to their own security and access control policies which may be context driven [Tri04, Jos05a]. Depending upon the type of workflow applications, several contextual parameters such as time, location, environment, agenda etc., may be considered and can pose substantial challenges in information security [Ant01, Ber99a, Wel03]. In particular, the resource access requirements of distributed workflows may conflict with the access control policies of service providers/domains [Sha05].

In order to develop secure distributed workflow applications, security assurance must be incorporated in the application design from the onset and such design must conform to the security requirements of all stakeholders. A key security requirement of any security critical system is accountability which entails that only authorized users or processes running on behalf of authorized users should be able to use the system's resources or functionalities [Lan01, Lam00, San94]. Therefore, access control that determines authorization of users, plays a critical role in establishing accountability for any system. In this paper, we focus on the authorization aspect of accountability in an autonomous multi-domain environment supporting distributed workflows, which need to be executed on a recurrent basis and require long term collaboration among domains. For verifying composibility of such workflows, their specifications need to be tested for conformance with the access control policies of all collaborating domains.

Examples of recurrent distributed workflow applications include: check clearance processing among banks, insurance claim processing, health-care administration, real-time process control systems, and distributed data processing for stream data warehouses [Gho93, Wod96, Ngu04, Jun04, Yu04, Tri03]. In all these applications, a predefined workflow specifies a logical sequence of activities or tasks that needs to be performed by collaborating and possibly autonomous domains. Some of these applications have strict deadlines for workflow completion which may not always be satisfied because of the time dependent access control policies of domains. These workflow applications are recurrent in a sense that they need to be invoked repeatedly after a fixed or variable time interval. For instance, the check clearance workflow among banks is invoked regularly to process a batch of check clearance requests [Gho93, Car88]. Similarly, the distributed data processing workflow for zero latency data stream warehouse is periodically invoked for mining the continuous data streams in near real-time [Ngu04]. For verifying secure composibility of such workflows, the following two questions need to be answered:

- Does the security and authorization policies of collaborating domains, support execution of the distributed workflow under the given timing constraints?
- What are the possible time instants at which the distributed workflow can be invoked recurrently?

The proposed composibility verification approach is designed to answer the above questions for a given distributed workflow specification. The domains to which the workflow tasks are assigned are considered to be autonomous with time dependent non-reentrant behavior [Agr82, Gac98]. The behavior of a domain is characterized as *reentrant* or *non-reentrant* based on the underlying software system enforcing a time-dependent access control policy. We use Generalized Temporal Role Based Access Control (GTRBAC) [Jos05a] model to specify the time dependent access control policy of a domain. In the software engineering terminology, a *non-reentrant* system does not allow its multiple simultaneous, interleaved, or nested invocations and only one instance of such system exists at any time [Gac98]. The *non-reentrant* behavior of a system is governed by its finite state model (FSM) and any interaction with such system has to be compatible with its current state [Gac98]. At any time, a domain can have only one instance of its GTRBAC policy against which all access requests are evaluated. In addition, the GTRBAC policy instance has a finite number of authorization states. Therefore, according to the above criterion a domain is a *non-reentrant* system.

For verifying workflow composibility, the distributed workflow specifications need to be analyzed for being consistent with the individual as well as with the collective behavior of collaborating domains. Accordingly workflow composibility verification entails two steps: i) verification of workflow specifications with respect to the FSM of individual domains, and ii) verification of dependencies among domains for execution of workflow tasks. These two steps can either be carried out separately in the above order, or can be performed simultaneously by using a unified global meta-policy that captures all intra-domain and inter-domain authorizations [Bel02, Sha05, She90]. The methodology proposed in this paper uses the two step verification approach and does not consider the meta-policy based approach for the following reasons:

- The unified global meta-policy is composed by integrating the access control policies of all collaborating domains; however, domains may not disclose their policies due to privacy concerns.
- More importantly, domains are autonomous in deciding when to join or leave the collaborative environment. Whenever a new domain joins the collaboration, the meta-policy needs to be reconfigured. Consequently, all the workflows verified with respect to the previous meta-policy need to be verified again. Such re-verification of existing workflows due to joining of new domains is not needed in the proposed methodology. The meta-policy is also reconfigured when any domain leaves the collaboration or changes/updates its access control policy. Again such reconfiguration of meta-policy triggers re-verification of all previously verified workflows including the ones that do not have any task assigned to the departing domains or domains that have changed/updated their policies. In

the proposed approach, a workflow is re-verified only if a domain participating in workflow execution updates its policy or leaves the collaboration.

Figure 1 depicts the proposed two step approach for verification of secure workflow composition. The approach relies on decomposition of a distributed workflow into domain-specific workflows called *projected workflows*. These projected workflows are verified by the respective domains in terms of the authorization and execution time requirements. After verification of projected workflows, the cross-domain dependencies amongst the workflow tasks performed by different collaborating domains are verified. The timing information computed in the projected workflow verification phase is used to determine an interleaving of projected workflow tasks that satisfies the cross-domain dependencies of the distributed workflow. This timing information is also used to determine a feasible schedule for the overall verified distributed workflow. For workflow composibility verification, we assume that the FSM of each domain's GTRBAC policy is given and the distributed workflow is specified using interaction model (IM).

Although the proposed verification approach is for secure workflow composibility, the approach is generic and can be applied to many distributed applications involving collaborations among *non-reentrant* and autonomous components. Examples of such applications include process control systems [Jaf91], mission planning and control in military systems [Coh02], real-time speech recognition systems [Erm80], and workflow-based production systems [Mue04]. The underlying verification problem in such applications is to determine whether or not a given configuration of *non-reentrant* components can support the functionality required by distributed applications.

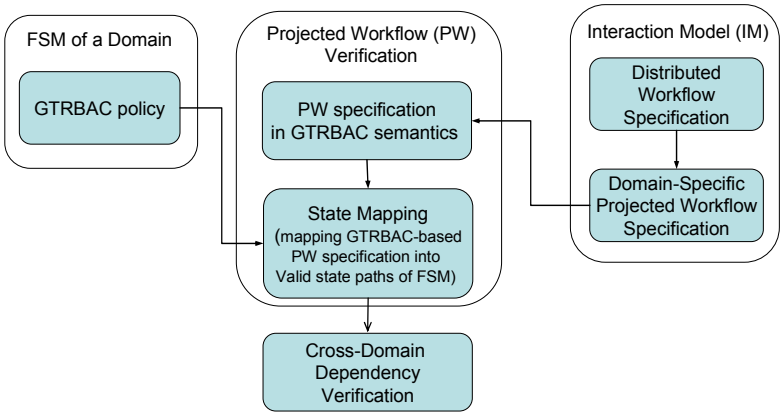



Figure 1. Overall process for workflow composibility.

The paper is organized in the following manner. Section 2 introduces the interaction model (IM) for distributed workflow specification and discusses how domain-specific projected workflows are created. Section 3 provides a brief overview of GTRBAC model and describes the state-based representation on GTRBAC policy. Section 4 presents a formal definition of secure workflow composibility and describes the proposed approach for verification of distributed workflows. Section 5 presents related work and Section 6 provides concluding remarks.

## 2 Interaction Model for Workflow Specification

For verifying workflow composibility, a formal and precise specification of the distributed workflow is needed. In particular, the specification should be able to capture the collaboration requirements among the domains performing the tasks of the distributed workflow. We use the term *component service* to refer to a task or set of tasks in a distributed workflow that can be executed by a domain independently. More precisely, a *component service* encapsulates a set of domain-specific tasks that are advertised to other interacting domains as a single capability/functionality of the domain.

Interaction models, such as Unified Modeling Language (UML) sequence diagrams [OMG03] and International Telecommunication Union (ITU) message sequence charts (MSC) [ITU96], have been widely used to model specifications of distributed workflows requiring communication among collaborating domains for service provisioning [Fos03, Kru04]. In this paper, we use UML 2.0 sequence diagrams to model the distributed workflow specification. A sequence diagram, shown in Figure 2, specifies the communication among the interacting entities as message exchanges. The vertical line in a sequence diagram represents time and is called the *lifeline* of the corresponding interacting entity. Message exchange between two entities is shown by an arrow from the sender to the receiver. The communication between the interacting entities can be either *synchronous* or *asynchronous*. In *synchronous* communication the sender blocks for the subsequent action to complete, whereas, there is no nesting of control in *asynchronous* communication. In this paper we consider all the message exchanges to be *synchronous* for the sake of simplicity.

In the following, we provide a formal definition of workflow sequence diagrams considered in this paper.

### 2.1 Workflow Sequence Diagram (WSD)

In WSD, we consider the interacting entities of a sequence diagram as interacting domains (IDs) defined in the following definition. In this definition, the incoming and outgoing messages at an ID corresponds to input and output events respectively.

*Definition 1 (ID). An Interacting Domain (ID) is a tuple {EV,≤, CS, T } where,*

*(a)* $EV = In \cup Out$ *is a set of events which are partitioned into input and output events.*

*(b)* $\leq \subseteq EV \times EV$ *is a partial ordering of events such that* $ev_i < ev_j \Rightarrow ev_i \; occurs \; before \; ev_j$

*(c)* $CS: \{c_1, c_2, \ldots c_r\}$ *is a set of component services offered by the ID*

*(d)* $T: In \rightarrow (CS \times 2^{Out})$ *maps the input event to the corresponding CS and set of output events.*

*Definition 2 (WSD). A Workflow Sequence Diagram (WSD) is a tuple WSD= {ID, TR} such that*:

*(a) ID= {$ID_1$, $ID_2$, $ID_3$ ,…….$ID_r$ } is a finite set of interacting domains $ID_i$={$EV_i$ ,≤$_i$, $CS_i$, $T_i$}* $1 \le i \le r, r \le d$ *where d is the total number of domains and $EV_i$ are disjoint sets of events. Without loss of generality that $ID_1$ always initiates the interaction.*

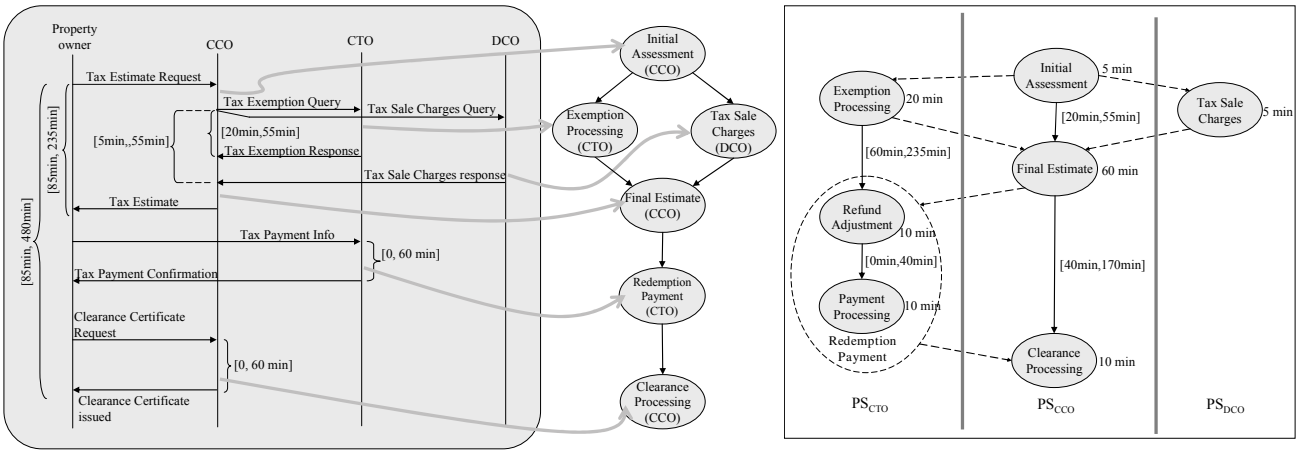
*(b) For a given $ID_j$, TR maps a pair of events to the minimum and maximum duration allowed between them. TR($ev_i$, $ev_j$)= [$d_l$,$d_u$] where $d_l$, $d_u \in Z^+$, $ev_i < ev_i$ and $i \neq j$ .*

The set ID in a WSD contains all the interacting domains that provide the required *component services* for workflow composition. The set of *component services* offered by an ID is specified in its definition. A *component service* is associated with one or more input events. An input event occurs with the arrival of an incoming message. The mapping function *T* in the ID definition maps the input event to a *component service* and a set of output events (output messages). The second element in the *WSD* tuple *TR* is a function that maps any pair of events ($ev_i$, $ev_j$) to a finite time interval. This interval specifies the minimum and maximum duration allowed between $ev_i$ and $ev_j$ provided that $ev_i$ occurs before $ev_j$. The WSD considered in this paper supports the notion of parallel interactions through concurrent message transmission as specified in UML 2.0 sequence diagrams [Pil05]. Such parallel interactions are needed to model the parallel invocation of *component service*s in different IDs. For instance, the concurrent messages "Tax Exemption Query" and "Tax Sale Charge Query" corresponds to parallel invocation of the *tax exemption processing* service in County Treasurer Office (CTO) and *tax sale charges processing* service in District Clerk Office (DCO).

*Example* 1: Figure 2(a) shows the WSD of a distributed workflow involving urgent processing of tax redemption request for delinquent real-estate property. The urgent processing entails that the entire business process of tax redemption be completed in one business day. The domains involved in provisioning of this distributed workflow include: property owner, County Clerk Office ($ID_{CCO}$), County Treasurer Office ($ID_{CTO}$), and District Clerk Office ($ID_{DCO}$) as shown in Figure 2(a). The distributed workflow of Figure 2(a) is initiated by the property owner by filing a tax estimate request for the delinquent property with $ID_{CCO}$. This request invokes the *initial assessment* service in $ID_{CCO}$. After completion of the *initial assessment*, the $ID_{CTO}$ and $ID_{DCO}$ are queried for *exemption processing* and *tax sale charges* for the given delinquent property index. Based on the exemption amount and tax sale charges returned by $ID_{CTO}$ and $ID_{DCO}$ respectively, the *final estimate* for the tax redemption amount is calculated and is submitted to the property owner. Upon receiving the redemption cost estimate, the property owner initiates the *payment processing* service for tax redemption with $ID_{CTO}$. After the *payment processing* is completed, the property owner requests for issuance of delinquent tax clearance certificate which launches the *clearance processing* service in $ID_{CCO}$.

The *component service*s associated with the events of the WSD of Figure 2(a) are shown in Figure 2(b). The time interval between events in the WSD of Figure 2(a) corresponds to the interval returned by the *TR* function for the corresponding events pair as specified in the WSD definition. For instance, the time interval [85min, 480min] between the tax exemption request, initiating the *tax redemption processing* workflow, and

the clearance certificate issued event implies that the distributed workflow must complete within 480 minutes (8 hours) relative to the initiation time of the workflow. The lower bound of 85 minutes implies that processing of this distributed workflow takes at least 85 minutes.




(a) (b) (c)

Figure 2. (a) WSD of a distributed workflow involving urgent processing of tax redemption request for delinquent real-estate property. (b) *component service*s required for performing tax redemption processing. (c) PW specification for each domain.

## 2.2 Domain-specific Projected Workflow Specification

The WSD specifies a high level description of the distributed workflow and considers the *component service*s as atomic operations provided by domains. However, a *component service* may encapsulate a workflow process comprising multiple tasks. For example the *redemption payment* service, shown in Figure 2(c) comprises two tasks, namely, *refund adjustment* and *payment processing*. The *refund adjustment* task precedes the *payment processing* task in the workflow associated with the *redemption payment* service. This low level description of the *component service* is specific to a domain and is not required for distributed workflow specification. However, as discussed later, such description is needed for composibility verification, which is performed for each domain separately. In the following we formally define a *component service* of an ID.

*Definition 4 (CS). For an ID, a component service* $(c \in CS)$ *is a tuple* $c = \{T_c, \leq, \lambda, \beta\}$ *where* $T_c$ *is a set of tasks included in the workflow of c, and* $\leq \subseteq T_c \times T_c$ *specifies partial ordering between the tasks such that* $\tau_1 \leq \tau_2$ $(\tau_1, \tau_2 \in T_c)$ *implies that* $\tau_1$ *precedes* $\tau_2$ *in the execution order.* $\lambda: T_c \rightarrow Z^+$ *is a function that maps a task to the time duration required for its completion.* $\beta: T_c \times T_c \rightarrow Z^+ \times Z^+$ $[d_l, d_u]$ *maps a task pair* $(\tau_1, \tau_2)$ *to an interval* $[d_l, d_u]$ $(d_l, d_u \in Z^+)$ *that specify the minimum and maximum duration between the completion and initiation of* $\tau_1$ *and* $\tau_2$ *respectively.*

Each task of the *component service* has certain authorization constraints specified in the corresponding domain's access control policy. In order to verify workflow composibility, we need to ensure that all the authorization constraints associated with each task of the *component service* are satisfied. As discussed in the

Introduction, each domain is autonomous and may not reveal its access control policy to other domains for security and privacy concerns. To facilitate workflow composibility verification with respect to the access control policy of each domain, a domain-specific projected workflow (PW) specification is generated from the WSD of the distributed workflow. The PW specification provides the following information related to the corresponding domain's involvement in the distributed workflow: i) *component service*s provided by the domain in the WSD, ii) temporal constraints between the *component service*s, and iii) the task workflow associated with each *component service* as defined above. We model the PW of a domain as a directed acyclic graph, which is constructed from the WSD and the *component service* specification. Each node of PW graph represents a task and an edge $(\tau_1, \tau_2)$ represents the precedence relationship between the tasks $\tau_1$ and $\tau_2$. The node of a PW graph is annotated with a non-negative integer specifying the time duration for completion of the corresponding task. This information is obtained from the task duration mapping function $\lambda$ given in the respective CS specification. Each edge in a PW graph is annotated with an interval that specifies the minimum and maximum duration between the completion and initiation of successive tasks connected by the edge. In case the successive tasks belong to the same CS the interval mapping function $\beta$ provides this information; otherwise, the interval is computed from the WSD specification.

Figure 2(c) shows the PW graph of $ID_{CTO}$, $ID_{CCO}$ and $ID_{DCO}$ for the tax redemption workflow described in Example 1. The dashed arrows in Figure 2(c) are not a part of any PW graph and are used to illustrate the temporal ordering of cross-domain *component service*s in the distributed workflow specification.

## 3 FSM of a domain's Access Control Policy

To analyze the consistency of the workflow specification against a domain's dynamic and *non-reentrant* behavior, a state based representation of the domain's access control policy is needed. In the following, we first provide an overview of the GTRBAC model for specification of time-dependent access control policies and then describe a finite state model (FSM) for state based representation of GTRBAC policies.

### 3.1 GTRBAC Model

GTRBAC is a temporal extension of the role-based access control (RBAC) model proposed by Sandhu *et. al.* in [San96]. RBAC consists of following four basic components: a set of user U, a set of roles R, a set of permissions P, and a set of sessions SE. A user is a human being or a process within a system. A role is a collection of permissions needed to perform a certain function or task. A permission is an access mode that can be exercised on a particular object or resource in the system. A session relates a user to possibly many roles and allows the user to access all permissions associated with such roles. A key aspect of the GTRBAC model is the notion of states of a role. In GTRBAC, a role can be in one of the three states: *disabled*, *enabled*, and *active*. A role is *enabled* if a user can access the permissions assigned to the role. An *enabled* role becomes active when a user accesses the permissions assigned to the role. By contrast, a *disabled role* cannot be activated by any user.

### 3.1.1 Preliminaries and Assumptions

A domain's GTRBAC policy specifies the authorizations for its *component services*. As mentioned in Section 2, a *component service* is essentially an encapsulation of one or more tasks with certain temporal and ordering constraints. At the interaction modeling level, a task is viewed as an operation on a resource by an authorized subject without considering who is authorized for the resource access and how such operation can be executed. In the GTRBAC model, a task can be represented as an activation of a particular role by an authorized user, where the role is a collection of permissions required to perform the requested operation on the underlying resource object(s) and the user corresponds to the subject executing the task. For establishing the semantics relationship between a *component service* and the underlying user-role activation in the GTRBAC policy, we define a function called *domain role mapper* (DRM) that maps each task of the *component service* to a set of user role activation pairs ($u$, $r$) such that each role $r$ in the pair ($u$, $r$) has all the relevant permissions required for processing of the task and the user $u$ is authorized for role $r$. Formally: $DRM(\tau) = \{(u, r) \mid u,$ *is authorized for* $r$ *and* $r$ *contains all permission required for processing* $\tau\}$.

As discussed in the Introduction, we are interested in verification of distributed workflow*s* that are executed recurrently. For supporting such workflows a domain's GTRBAC policy must allow access to its roles on a recurrent basis. Therefore, we consider only those roles that can be accessed infinitely often and have a periodic enabling time. For instance, a *tax filing* role in the tax payment workflow is enabled daily from 9:00 am to 5:00pm and can be accessed any time within its enabling interval. GTRBAC allows specification of periodic time intervals for various role-related events including role enabling. These periodic intervals are specified using periodic expressions [Ber99b, Nie92]. A periodic expression is used to define an infinite set of periodic intervals. The periodic time uses the notion of calendar defined as a countable set of contiguous intervals [Ber99b]. We consider a set of calendars with granularities in *minutes*, *hours*, *days*, *weeks*, and *months*. A subcalendar relation can be established among these calendars. Given two calendars $Cal_1$ and $Cal_2$, $Cal_1$ is said to be a subcalendar of $Cal_2$, written as $Cal_1 \subseteq Cal_2$, if each interval of $Cal_2$ is covered by a finite number of intervals of $Cal_1$.

A periodic expression is defined as: $PE = \sum_{i=1}^{n} O_i Cal_i \triangleright x.Cal_d$ , where $Cal_d$, $Cal_1$, $Cal_2,\ldots,Cal_n$ are calendars and $O_n = all$, $O_i \in 2^{\mathbb{N}} \cup \{\text{all}\}$, $Cal_{i-1} \subseteq Cal_i$, and $x \in \mathbb{N}$. The symbol $\triangleright$ separates the first part of the periodic expression that distinguishes the set of starting points of the intervals, from the specification of the duration of each interval in terms of the calendar $Cal_d$. For example, $\{all.days, \{9, 15, 23\}.Hours, \{20, 50\}.Minutes \triangleright 15.Minutes\}$ represent the set of intervals {[09:20, 09:35], [09:50, 10:05], [15:20, 15:35], [15:50, 16:05], [23:20, 23:35], [23:50, 23:59], [00:00, 00:05], [09:20, 09:35], [09:50, 10:05]…..}.

We assume that the periodic expressions corresponding to enabling of each role are specified using the same number of calendars. For instance, if $PE_1 = \sum_{i=1}^{n} O_i Cal_i \triangleright x.Cal_d$ and $PE_2 = \sum_{j=1}^{m} O_j Cal_j \triangleright y.Cal_d$ denote the periodic expressions for enabling of any two roles then $Cal_n = Cal_m$. If $Cal_m < Cal_n$, the left slicing operation defined in [Nie92] can be used to expand $PE_2$ to $Cal_n$ or vice versa. The duration of the calendar $Cal_n$ in terms of the basic calendar $Cal_1$ is given by *duration*($Cal_n$/$Cal_1$). Let $I_n$ = [0, *duration*($Cal_n$/$Cal_1$)] denote the interval associated with a period of calendar $Cal_n$. We assume that the calendar $Cal_n$ is the smallest calendar that contains the enabling intervals of all roles of the given GTRBAC policy. Therefore, $I_n$ corresponds to the smallest interval that covers the enabling interval of all roles in one calendar period. We denote the set of all intervals of a periodic expression *PE* that are contained in $I_n$ by $\Gamma(I_n, PE)$. For example, for $I_n$ = [0, 1440 minutes (1 day)] and *PE* = {*all.days*, {9, 15, 23}.*Hours*, {20, 50}.*Minutes* ▷ 15.*Minutes*}, $\Gamma(I_n, PE)$ = {[09:20, 09:35], [09:50, 10:05], [15:20, 15:35], [15:50, 16:05], [23:20, 23:35], [23:50, 23:59], [00:00, 00:05]}. For computing $\Gamma(I_n, PE)$, we divide any interval $I$ = [a , b] of a PE that overlaps with $I_n$ but is not fully contained in $I_n$, into two intervals $I_1$ = [$a$, *duration*($Cal_n$/$Cal_1$)] and $I_2$ = [*0*, b - *duration*($Cal_n$/$Cal_1$)].

Periodic intervals can be specified for various constraints such as, role enabling, role assignment, and role activation. However, for simplicity we consider periodic intervals for role enabling events only. Given a role set R and an interval $I_n$, the enabling intervals of all roles in R that are contained in $I_n$ is denoted by the set $EI_R$.

$$EI_R = \bigcup_{r \in R} \Gamma(I_n, PE_r) = \bigcup_{r \in R} \{[a_r, b_r]\},$$

where, $PE_r$ is the periodic expression for enabling of role $r$. We assume that each role has only one enabling interval in $EI_R$. If a role $r$ has multiple enabling intervals, say $m$, then we create roles $r_1$, $r_2$,…, $r_m$, one for each of the $m$ intervals. Each role $r_j$ is similar to $r$ in user-role assignment, role permission assignment, SoD, and trigger constraints.

### 3.1.2 GTRBAC Policy Specification

In this section, we discuss the syntax and semantics of various GTRBAC constraints used to specify the time dependent access control policies of domains. The GTRBAC constraints considered in this paper can be divided in to six types: i) user-role assignments and role-permission assignments, ii) periodicity constraints on role enabling, iii) role activation constraints, iv) run-time events, v) triggers, and vi) separation of duty (SoD) constraints. These constraints are summarized in Table 1.

Table 1. GTRBAC Constraints

| $r \in$ R, $u \in$ U, $p \in$ P, tg $\in$ TRG<br>R is a set of roles, U is a set of users, P is a set of permissions, and TRG is a set of Triggers | | | |
|---|---|---|---|
| Constraints | | Expression | Semantics |
| User-role assignment | | (assign$_U$ $r$ to $u$ ) | Role $r$ is assigned to user $u$ |
| Role-Permission assignment | | (assign$_U$ $p$ to $r$ ) | Permission $p$ is assigned to role $r$ |
| Periodicity constraint on role enabling | | ($PE_r$ enable $r$ ) | Role $r$ is periodically enabled during the intervals contained in the periodic expression $PE_r$ |
| Duration constraint on role activation | | ([$d_r^{min}$, $d_r^{max}$] active $r$) | The activation duration of role r in any session must be greater than or equal to $d_r^{min}$, and less than or equal to $d_r^{max}$. |
| Run-time request | | (activate/deactivate $r$ for $u$) | User's/administrator's request for role activation or deactivation. |
| Trigger | | $ev, sp_1, ...sp_k \rightarrow ev'$ | An event $ev$ must be immediately followed by $ev'$ provided that all status predicates $sp_1, ...sp_k$ hold at the time of the occurrence of $ev$. |
| SoD | Role-specific | $\forall r, r' \in$ R-SoD($u$), u-active($u$, $r$) $\Rightarrow \neg$u-active($u$, $r'$) | R-SoD($u$) is a set of conflicting roles for user $u$, i.e., $u$ can activate at most one role in R-SoD($u$) at any given time. |
| | User-Specific | $\forall u, u' \in$ U-SoD($r$), u-active($u$, $r$) $\Rightarrow \neg$u-active($u'$, $r$) | U-SoD($r$) is a set of conflicting users for role $r$, i.e., $r$ can be activated by at most one user in U-SoD($r$) at any given time. |

The user-role and role-permission assignment expressions, listed in Table 1, specify the authorizations of users over the GTRBAC roles and the underlying resources. For simplicity, we do not consider any temporal and periodicity constraints on user-role and role permission assignments. Omission of these constraints does not restrict the expressiveness of the GTRBAC model considered in this paper, as the temporal constraints on user-role and role permission assignments can be specified using role enabling constraints [Jos05b]. The periodicity constraints on role enabling/disabling are specified using periodic expressions as discussed earlier. The role activation constraint, listed in row 3 of Table 1, specifies a lower and upper bound on the activation duration of a given role by any user. Accordingly, the activation duration of a role in any session must be greater than or equal to $d_r^{min}$ and less than or equal to $d_r^{max}$, where $d_r^{max} \geq d_r^{min} > 0$. The original GTRBAC model does not constrain the activation of a role to a minimum duration. However for state-based analysis of a GTRBAC policy, we require that a role be activated for a finite number of times in any finite time interval. To satisfy this requirement, we have introduced the minimum activation duration constraint for each role.

The run-time events allow an administrator or a user to request the activation or deactivation of a role. GTRBAC triggers are used to specify the dependence relationship among events. The expression for a trigger considered in this paper has the following form: $ev, sp_1, ...sp_k \rightarrow ev'$, where $ev$ is a simple event expression and $sp_i$s are GTRBAC status predicates listed in Table 2. The event $ev$ in the body of the trigger is called the triggering event and $ev'$ is called the triggered event. We consider the triggered event $ev$' to be a role deactivation event. Note that the original GTRBAC model allows $ev'$ to be role enabling or disabling event [Jos05a]. However, in this paper we have assumed that roles are automatically enabled during the

specified time intervals. Therefore, defining triggers for role enabling or disabling event will violate this assumption.

Separation of duty (SoD) constraints, listed in Table 1, are used to prevent conflicting users from accessing same role concurrently or to prohibit conflicting roles from being accessed by same user at the same time. Although SoD constraints can be specified for user-role assignment, role enabling, and role activation, we consider SoDs that are specific to role activations only. We assume that the user-role assignment remains fixed throughout the policy life time and no periodic or temporal constraint is defined on such assignments, therefore, assignment-specific SoDs are not considered in this paper.

Table2. Event and status predicates

| $r \in R, u \in U, p \in P, tg \in TRG$<br>R is a set of roles, U is a set of users, P is a set of permissions, and TRG is a set of Triggers | | | |
|---|---|---|---|
| Simple Event | | Status Predicate | Semantics |
| `enable` *r* | | `Ur-assigned`(*u*, *r*) | *u* is assigned to *r* |
| `disable` *r* | | `Pr-assigned`(*p*, *r*) | *p* is assigned to *r* |
| `activate` *r* for *u* | | `r-enabled`(*r*) | *r* is enabled |
| `de-activate` *r* for *u* | | `r-active`(*r*) | r is active in at least one user's session |
| | | `u-active`(*u*, *r*) | *r* is active in *u*'s session |
| | | `trg-enabled`(*tg*) | Trigger *tg* is enabled, i.e., the event *ev* defined in the body of the trigger *tg* has occurred and the status predicates hold. |

*Example* 2: Table 3 shows the GTRBAC policies of domains $ID_{CTO}$ and $ID_{CCO}$ collaborating for tax collection and payment processing. The roles of the $ID_{CTO}$ include Tax Exemption Processor (TEP), Tax Refund Processor (TRP), and Tax Payment Processor (TPP). The user role assignments of the GTRBAC policy of $ID_{CTO}$ are as follows: $u_1$ is assigned all three roles, $u_2$ is assigned the TPP role, and $u_3$ is assigned the TRP role. TEP is authorized for accessing the tax exemption records for verification and approval of *exemptions* claimed by property owners. TRP performs tax *refund adjustment* in the tax bills due for payment. For this purpose, TEP has appropriate authorization for accessing tax billing and refund records. TPP *processes payments* of adjusted tax bills by property owners. For processing such payments, TPP is assigned a read permission on tax billing records and read/write permission on tax payment records as shown in Table 3. TEP is enabled daily from 9:00am to 4:00 pm, while TRP and TPP are enabled from 10:00am to 2:00pm every day. For security reasons, $ID_{CTO}$ does not allow a single user to perform both *exemption processing* and *payment processing* for the same property index. This constraint is defined as a role-specific separation of duty (SoD) constraint between TEP and TRP roles, prohibiting any user (in this case $u_1$) to activate both roles TEP and TRP simultaneously.

The GTRBAC policy of the domain $ID_{CCO}$ includes four roles, namely: Tax Assessment Processor (TAP) and Delinquent Tax Processor (DTP). TAP is authorized to access tax delinquency and property ownership records for performing an *initial assessment* of tax redemption cost. DTP is responsible for preparing the *final estimate* for tax redemption. In addition, DTP also performs *clearance processing*. Both TAP and DTP can be enabled from 8:00am to 12:00pm and from 2:00pm to 5:00pm. Since we have assumed that a role can

have only one enabling interval in a single calendar period, therefore both TAP and DTP are split into two roles, namely: $TAP_1$, $TAP_2$, $DTP_1$, and $DTP_2$. $TAP_1$ and $DTP_1$ are assigned the enabling interval of [8:00am, 12:00pm], whereas, $TAP_2$ and $DTP_2$ are assigned the enabling interval [2:00pm, 5:00pm].

Table 3. GTRBAC Policies of CTO and CCO domains

| | | | |
|---|---|---|---|
| County Treasurer Office ($ID_{CTO}$) | | | |
| User-role assignment | 1 | $assign_U$ $u_1$ to {TEP, TPP}; $assign_U$ $u_2$ to TPP; $assign_U$ $u_3$ to TRP | |
| Role-permission assignment | 2 | $assign_P$ $p_1$(Tax Exemption Records, {read,write,approve}) to TEP;<br>$assign_P$ $p_2$(Tax Refund Records, {read,write,approve}) to TRP;<br>$assign_P$ $p_3$(Tax Payment Records, {read,write,approve}) to TPP;<br>$assign_P$ $p_4$(Tax Billing Records, {read,write,approve}) to TRP and TEP;<br>$assign_P$ $p_5$(Tax Billing Records, {read) to TPP and TRP | |
| Periodicity constraints on role enabling | 3 | $PE_{TEP}$: all.Days+ 10.Hours+1.Minutes⊳ 420.Minutes; $\Gamma(I_n, PE_{TEP})$ = [09:00, 16:00]<br>$PE_{TPP}$: all.Days+ 11.Hours+1.Minutes⊳ 240.Minutes; $\Gamma(I_n, PE_{TEP})$ = [10:00, 14:00]<br>$PE_{TRP}$: all.Days+ 11.Hours+1.Minutes⊳ 240.Minutes; $\Gamma(I_n, PE_{TEP})$ = [10:00, 14:00] | ($PE_{TEP}$, enable TEP)<br>($PE_{TPP}$, enable TEP)<br>($PE_{TRP}$, enable TEP) |
| Role-activation constraint | 4 | ([120min, 420min] active TEP); ([240min, 240min] active TPP); ([240min, 240min] active TRP) | |
| Separation of Duty | 5 | U-SoD(TPP) = {$u_1$, $u_2$} | |
| County Clerk Office ($ID_{CCO}$) | | | |
| User-role assignment | 6 | $assign_U$ $u_4$ to {$TAP_1$, $TAP_2$}; $assign_U$ $u_5$ to {$DTP_1$, $DTP_2$} | |
| Role-permission assignment | 7 | $assign_P$ $p_6$(Tax Delinquency Records, {read}) to {$TAP_1$, $TAP_2$};<br>$assign_P$ $p_7$(Property ownership Records, {read}) to {$TAP_1$, $TAP_2$};<br>$assign_P$ $p_8$(Tax Exemption Records, {read}) to {$DTP_1$, $DTP_2$};<br>$assign_P$ $p_9$(Tax Sale Records, {read}) to {$DTP_1$, $DTP_2$};<br>$assign_P$ $p_{10}$(Redemption Invoice, {read,write,approve}) to {$DTP_1$, $DTP_2$} | |
| Periodicity constraints on role enabling | 8 | $PE_{TAP1}$: all.Days+ 9.Hours+1.Minutes⊳ 240.Minutes; $\Gamma(I_n, PE_{TAP1})$ = [08:00, 12:00]<br>$PE_{TAP2}$: all.Days+ 15.Hours+1.Minutes⊳ 180.Minutes; $\Gamma(I_n, PE_{TAP2})$ = [14:00, 17:00]<br>$PE_{DTP1}$: all.Days+ 9.Hours+1.Minutes⊳ 240.Minutes; $\Gamma(I_n, PE_{DTP1})$ = [08:00, 12:00]<br>$PE_{DTP2}$: all.Days+ 15.Hours+1.Minutes⊳ 180.Minutes; $\Gamma(I_n, PE_{DTP2})$ = [14:00, 17:00] | ($PE_{TAP1}$, enable $TAP_1$)<br>($PE_{TAP2}$, enable $TAP_2$)<br>($PE_{DTP1}$, enable $DTP_1$)<br>($PE_{DTP2}$, enable $DTP_2$) |
| Role-activation constraint | 9 | ([240min, 240min] active $TAP_1$); ([240min, 240min] active $DTP_1$); ([180min, 180min] active $TAP_2$); ([180min, 180min] active $DTP_2$) | |

## 3.2 State-Based Representation of GTRBAC Policy

We model the GTRBAC policy of a domain as a timed graph introduced by Alur *et. al.*[Alu93, Alu94]. Timed graphs have been widely used to characterize behavior of real-time systems having finite number of states. A timed graph is a directed graph consisting of a finite set of nodes, a finite set of edges, and a finite set of real-valued clocks. The following definition characterizes the state space of the FSM of GTRBAC policy.

*Definition* 5 [GTRBAC Timed graph]: A GTRBAC timed graph is represented by a tuple TG = <S, SP, $\mu$, $s_0$, E, C, $c_0$, $b_{max}$, $\gamma$, $\delta$ >, where

- S is a finite set of nodes representing GTRBAC states.
- SP denote the set of GTRBAC status predicates. SP={r-enabled(r)| r ∈ R}∪{u-active(u, r)| u∈U, r∈R, and u-assigned(u, r)}∪{trg-enabled(tg)| tg is a trigger in GTRBAC policy}.
- $\mu$: S → A ⊆ $2^{SP}$ is a labeling function assigning to each state the set of status predicates that are true in that state. Where, A is the maximal subset of $2^{SP}$ such that predicate assignment $a$ ∈ A satisfies all the GTRBAC constraints listed in Table 4.

- $E \subseteq S \times S$ is a set of edges. The edges represent the events causing the domain to move from one GTRBAC state to another.
- $s_0 \in S$ is the initial state and $s_{reset} \in S$ is the calendar clock reset state. In state $s_0$ and $s_{reset}$ all roles are disabled. For all state $s \in S - \{s_0, s_{reset}\}$, $(s, s_{reset}) \notin E$.
- C is a finite set of clocks.
- $c_0$ is a calender clock which is reset with the occurrence of clock reset event represented by the edge from $s_0$ to $s_{reset}$.
- $b_{max} = \max_{r \in R} \{b_r\}$, where $b_r$ is the end point of the interval during which role *r* is enabled.
- $\gamma$ is a function labeling each edge with an enabling condition of the form $(d_1 \le c_0 \le d_2) \wedge_{x \in C'} (d_{1x} \le x \le d_{2x})$, where $C' \subseteq C$ and $d_1, d_2, d_{1x}, d_{2x} \in Z^+$ with $d_1 \le d_2 \le b_{max}$ and $d_{1x} \le d_{2x} < b_{max}$. For the edge $e_{reset}$ from $s_0$ to $s_{reset}$, $\gamma(e_{reset}) = b_{max} \le c_0 \le b_{max}$, and for the edge $e_0$ from $s_{reset}$ to $s_0$, $\gamma(e_0) = 0 \le c_0 \le 0$.
- $\delta : E \rightarrow 2^C$ is a function mapping an edge to a set (possibly an empty set) of clocks that are reset with the edge. The function $\delta$ maps the edge $e_{reset}$ from $s_0$ to $s_{reset}$ to $c_0$, i.e., $\delta(e_{reset}) = c_0$.

A node in a GTRBAC timed graph models the access control state of a domain characterized by the status predicates true in that state. All states in S satisfy the GTRBAC policy constraints including separation of duty constraints, dependence constraint between role enabling and role activation, dependence constraint between role assignment and activation, and trigger enabling constraint. These constraints are listed Table 4. Edges in the GTRBAC timed graph represent the state transition events, which are listed in Table 3. Each edge is labeled with an enabling condition defined using clock values. At any point in time, the domain can make a transition from its current state $s_i$ to a next state $s_j$, if the enabling condition associated with the edge $(s_i, s_j)$ is satisfied by the current values of clock. A clock can be reset with any state transition. At any instant, the value of a clock is equal to the time elapsed since the last time the clock was reset. Each edge in the GTRBAC timed graph is mapped to a set (possibly an empty set) of clocks that are reset when the corresponding state transition event occurs. In states $s_0$ and $s_{reset}$ all roles are disabled. The state $s_{reset}$ is visited when no role can be enabled during the current calendar period. By visiting state $s_{reset}$, the calendar clock $c_0$ of a domain is reset/initialized to the starting point of next calendar period in which the enabling and activation of roles follow the pattern of previous calendar periods.

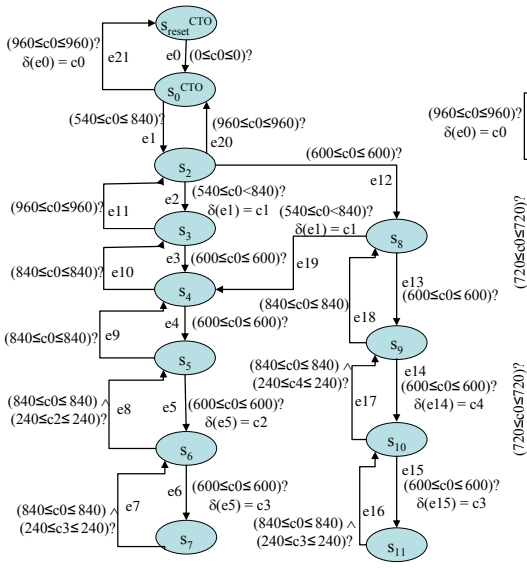
The procedure for generating the FSM of a GTRBAC policy in a timed graph representation is depicted in Figure 3. This procedure first generates the state space of the given GTRBAC policy by considering all valid status predicate assignments that satisfy the GTRBAC constraints listed in Table 4. After generating the state space, the state transitions in the GTRBAC timed graph are defined by creating the edge set E. For all pairs of GTRBAC states $s_i$ and $s_j$, an edge is created from $s_i$ to $s_j$ only if there exists a GTRBAC event $ev_{ij}$

such that $s_i$ satisfies all the precondition of $ev_{ij}$ and $s_j$ satisfies the post conditions of $ev_{ij}$. Next the edges in the set E are labeled with appropriate enabling conditions and clock reset function.

GTRBAC-FSM
INPUT: GTRBAC Policy Instance
OUTPUT: Timed graph

1. Let SP={`r-enabled`($r$)| $r \in$ R}$\cup${`u-active`($u$, $r$)| $u \in$U, $r \in$R, and `u-assigned`($u$, $r$)}$\cup${`trg-enabled`($tg$)| $tg$ is a trigger in GTRBAC policy}
2. Let A be a maximal subset of $2^{SP}$ such that each $a \in$ A satisfies all the GTRBAC constraints listed in Table 4.
3. Generate a set of states S such that |S| = |A| + 1.
4. Create a one-to-one mapping $\mu$ from states in the set S-$\{s_{reset}\}$ to predicate assignments in the set A. Name the state that is mapped to an assignment in which all roles are disabled as $s_0$. Let $\mu(s_{reset}) = \mu(s_0)$.
5. For each pair of states $s_i$ and $s_j$, if there is a GTRBAC event $ev$ that can cause state transition from $s_i$ and $s_j$, then create an edge $e_{ij}$ from $s_i$ and $s_j$.
   a. If $ev$ is an enabling event of a role $r$ then define the following mapping:
      $\gamma(e_{ij}) = a_r \le c_0 \le a_r$
   b. If $ev$ is a disabling event of role $r$ then define the following mapping:
      $\gamma(e_{ij}) = (b_r \le c_0 \le b_r)$
   c. If $ev$ an activation event of role $r$ by user $u$ then define the following mappings:
      $\gamma(e_{ij}) = a_r \le c_0 \le (b_r - d_r^{min})$
      $\delta(e_{ij}) = c_{ur}^{act}$, where, $c_{ur}^{act} \in C$ is a clock that measure the time elapsed since activation of role $r$ by $u$.
   d. If $ev$ is a deactivation event of role $r$ by user $u$ then define the following mapping:
      $\gamma(e_{ij}) = (a_r + d_r^{min} \le c_0 \le b_r) \wedge (d_r^{min} \le c_{ur}^{act} \le d_r^{max})$
   e. For each GTRBAC trigger $tg$: $ev_1$, $sp_1, \ldots sp_k \rightarrow$ `deactivate` $r$ for $u$, such that t-enabled($tg$) $\in \mu(s_i)$, t-enabled($tg$) $\notin \mu(s_j)$, and ($ev$ == `deactivate` $r$ for $u$), then $\gamma(e_{ij}) = et(s_i)$.
6. Delete all edges $e_{ij}$ corresponding to some event $ev$, if there exists a GTRBAC trigger $tg$: $ev_1$, $sp_1, \ldots sp_k \rightarrow$ `deactivate` $r$ for $u$, such that one of the following holds:
   a. `t-enabled`($tg$) $\in \mu(s_i)$ and `t-enabled(tg)` $\in \mu(s_j)$.
   b. `t-enabled(tg)` $\in \mu(s_i)$ and ($ev \neq$ `deactivate` $r$ for $u$).
   c. ($ev \neq ev_1$) and `t-enabled(tg)` $\in \mu(s_j)$.
7. Create an edge $e_0$ from $s_{reset}$ to $s_0$ and define the following mapping:
   $\gamma(e_0) = 0 \le c_0 \le 0$
8. Create an edge $e_{reset}$ from $s_0$ and $s_{reset}$ and define the following mapping:
   $\gamma(e_{reset}) = b_{max} \le c_0 \le b_{max}$
   $\delta(e_{reset}) = c_0$

Figure 3. Procedure for generating the timed graph of a GTRBAC policy




| States | *enable*(TEP) | *enable*(TPP) | *enable*(TRP) | *u-activee*($u_1$,TEP) | *u-activee*($u_1$,TPP) | *u-activee*($u_2$,TPP) | *u-activee*($u_3$,TRP) |
|---|---|---|---|---|---|---|---|
| $s_0^{CTO}$, $s_{reset}^{CTO}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $s_2$ | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| $s_3$ | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| $s_4$ | 1 | 1 | 0 | 1 | 0 | 0 | 0 |
| $s_5$ | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| $s_6$ | 1 | 1 | 1 | 1 | 0 | 1 | 0 |
| $s_7$ | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| $s_8$ | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| $s_9$ | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| $s_{10}$ | 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| $s_{11}$ | 1 | 1 | 1 | 0 | 1 | 0 | 1 |

(c)

| States | *enable*($TAP_1$) | *enable*($DTP_1$) | *enable*($TAP_2$) | *enable*($DTP_2$) | *u-activee*($u_4$,$TAP_1$) | *u-activee*($u_5$,$DTP_1$) | *u-activee*($u_4$,$TAP_2$) | *u-activee*($u_5$,$DTP_2$) |
|---|---|---|---|---|---|---|---|---|
| $s_0^{CCO}$, $s_{reset}^{CCO}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $s_{20}$ | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $s_{21}$ | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| $s_{22}$ | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| $s_{23}$ | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| $s_{24}$ | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| $s_{25}$ | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| $s_{26}$ | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| $s_{27}$ | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |

(d)

| Edges | Event | Edges | Event | Edges | Events |
|---|---|---|---|---|---|
| e0 | New Period | e10 | *disable* TPP | e26 | *enable* $DTP_1$ |
| e1, | *enable* TEP | e11 | *deactivate* TEP for $u_1$ | e27 | *activate* $DTP_1$ for $u_5$ |
| e2,e19 | *activate* TEP for $u_1$ | e14 | *activate* TPP for $u_1$ | e28 | *disable* $DTP_1$ |
| e3,e12 | *enable* TPP | e17 | *deactivate* TPP for $u_1$ | e29 | *disable* $TAP_1$ |
| e4,e13 | *enable* TRP | e20 | *disable* TEP | e30 | *enable* $TAP_2$ |
| e5 | *activate* TPP for $u_2$ | e21 | Reset event for c0 in CTO | e31 | *activate* $TAP_2$ for $u_4$ |
| e6,e15 | *activate* TRP for $u_3$ | e22 | New Period | e32 | *enable* $DTP_2$ |
| e7,e16 | *deactivate* TRP for $u_3$ | e23 | Reset event for c0 in CTO | e33 | *activate* $DTP_2$ for $u_5$ |
| e8 | *deactivate* TPP for $u_2$ | e24 | *enable* $TAP_1$ | e34 | *disable* $DTP_2$ |
| e9,e18 | *disable* TRP | e25 | *activate* $TAP_1$ for $u_4$ | e35 | *disable* $TAP_2$ |

(e)

Figure 4. (a) Timed graph of the GTRBAC policy of $ID_{CTO}$. (b) Timed graph of the GTRBAC policy of $ID_{CCO}$. (c) GTRBAC status predicate assignment to the states of Figure 4(a). (d) GTRBAC status predicate assignment to the states of Figure 4(b). (e) Events associated with the edges of Figures 4(a) and 4(b).

The timed graph of the GTRBAC policy of the $ID_{CTO}$ generated by this procedure is shown in Figure 4(a). The initial state of this timed graph is $s_0^{CTO}$ and the calendar clock reset state is $s_{reset}^{CTO}$. The status predicates that are true in the GTRBAC states of Figure 4(a) are tabulated in Figure 4(c). The events corresponding to the edges of Figures 4(a) and 4(b) are listed in Figure 4(e). Each edge is labeled with an enabling condition defining the timing constraints for the corresponding GTRBAC event. For instance, the edge e2, representing the event *activate* TEP for $u_1$, is labeled with the enabling condition '$540 \leq c_0 \leq 840$'. This enabling condition implies that the TEP role can be activated by $u_1$ from the state $s_2$ within an interval of [540, 840] minutes. This interval is defined with respect to the calendar clock $c_0$ which is initialized (reset) in state $s_{reset}^{CTO}$.

Table 4. Constraints on all valid status predicate assignment to GTRBAC states

| SP={r-enabled($r$)\| $r \in$ R}$\cup${u-active($u$, $r$)\| $u \in$U, $r \in$R, and u-assigned($u$, $r$)}$\cup${trg-enabled($tg$)\| $tg$ is a trigger in GTRBAC policy}<br>$\forall a \in$ A such that A is a maximal subset of $2^{SP}$, the following constraints must be satisfied: | | |
|---|---|---|
| | Constraints | Meaning |
| 1 | u-active($u$, $r$) $\in a \Rightarrow$ r-enabled($r$) $\in a$ | A disabled role cannot be activated by any user in any GTRBAC state. |
| 2 | $\forall r$, $r$' $\in$ R such that $\Gamma(I_n, PE_r) \cap \Gamma(I_n, PE_{r'}) = \phi$,<br>r-enabled($r$) $\in a \Rightarrow$ r-enabled($r$') $\notin a$ | Two roles with disjoint enabling intervals cannot be enabled simultaneously in any GTRBAC state. |
| 3 | $\forall r$, $r$' $\in$ R such that $\Gamma(I_n, PE_r)=[a_r, b_r]$, $\Gamma(I_n, PE_{r'}) = [a_{r'}, b_{r'}]$,<br>and $a_{r'} \le a_r \le b_r \le b_{r'}$,<br>r-enabled($r$) $\in a \Rightarrow$ r-enabled($r$') $\in a$ | If the enabling interval of $r$ is contained in the enabling interval of $r$', then in any GTRBAC state in which $r$ is enabled, $r$' is also enabled. |
| 4 | $\forall r, r'$ $\in$ R-SoD($u$),<br>u_active($u$, $r$) $\in a \Rightarrow$ u_active($u$, $r$') $\notin a$ | In any GTRBAC state, a user $u$ can activate at most one role in R-SoD($u$). |
| 5 | $\forall u, u'$ $\in$ U-SoD($r$),<br>u_active($u$, $r$) $\in a \Rightarrow$ u_active($u'$, $r$) $\notin a$ | In any GTRBAC state, at most one user in the set U-SoD($r$) can activate $r$. |
| For any trigger $tg$: $ev_1, sp_1, \ldots sp_k \rightarrow$ deactivate $r$ for $u$ | | |
| 6 | trg-enabled($tg$) $\in a \Rightarrow \bigwedge_{i=1}^{k} sp_i \in a \wedge$<br>u-active($u$, $r$) $\in a \wedge$[ (u-active($u'$, $r'$) $\in a$ if $ev_1$==activate $r$' for $u$') $\vee$ (u-active($u'$, $r'$) $\notin a$ if $ev_1$==deactivate $r$' for $u$') $\vee$ (r-enable($r'$) $\in a$ if $ev_1$==enable $r'$) $\vee$ (r-enable($r'$) $\notin a$ if $ev_1$==disable $r'$)] | Trigger $tg$ is enabled only if the event $ev_1$ occurs and at the time of occurrence of $ev_1$, all status predicates defined in the triggers body hold.<br><br>After enabling of trigger $tg$, the only event that can occur is deactivate $r$ for $u$. |

### 3.3 Definition of State Path and Timing Constraints

In this section, we define state path and the state timing constraints. These timing constraints are used to determine the composibility of a given distributed workflow with respect to the FSM of domains as discussed in Section 4.

*Definition* 6 *(State Path)*: *A state path $\pi$ is a sequence $s_1.e_1.s_2.e_2\ldots\ldots,e_{n-1}.s_n$, $n > 0$, such that the symbol $s_i$ $(1 \le i \le n)$ in path $\pi$ denotes a GTRBAC state, and the symbol $e_j$ $(1 \le j \le n-1)$ denotes an edge in the timed graph of GTRBAC policy. The edge $e_j$ represents the event that causes a transition in the GTRBAC system from state $s_j$ to $s_{j+1}$.*

*Definition 7 (State Entry Time)*: *The time instant at which a GTRBAC state, say $s_j$, can be visited is called the entry time of state $s_j$ and is denoted by $et(s_j)$. The entry time of a state is measured relative to the domain's calendar clock $c_0$, which is initialized and reset in state $s_{\text{reset}}$ only.*

For computing $et(s_j)$, we need to determine the enabling and activation times of all roles that are enabled and active in state $s_j$. Let $R_j^{en}$ and $R_j^{act}$, respectively, denote the set of roles that are enabled and active in state $s_j$. The following constraint defines an upper and lower bound on value of $et(s_j)$.

$$\max_{r \in R_i^{en}} \{a_r\} \le et(s_j) \le \max_{r'' \in R_i^{act}} \{b_{r''} - d_{r''}^{\min}\},$$

where, $[a_r, b_r]$ is the enabling interval of role $r$, and $d_r^{min}$ is the minimum duration for which $r$ can be activated by any user.

*Definition 8 (State Residence Time)*: *The time a domain stays in a particular GTRBAC state, say $s_j$, in a state path, say $\pi$, is called the residence time of state $s_j$ in $\pi$.*

Let $t^{\pi}_{sj}$ denote the residence time of state $s_j$ in path $\pi$: $s_1.e_1\ldots\ldots e_{j-1}s_j.e_js_{j+1}\ldots,e_{n-1}.s_n$. Suppose $\gamma(e_j) = \bigwedge_{x\in C'} d_{1x} \le x \le d_{2x}$ is the enabling condition for the event represented by edge $e_j$, where $C' \subseteq C$ and $c_0 \in C'$. For a clock $x \in C'$, let $e_{j\text{-}kx}$ be an edge in $\pi$ such that $x \in \delta(e_{j\text{-}kx})$, (i.e., clock $x$ is reset at the edge $e_{j\text{-}kx}$) and there is no other edge $e_p$ between $e_{j\text{-}kx}$ and $e_j$ in $\pi$ for which $x \in \delta(e_p)$. The following inequalities provides a bound on the residence time $t^{\pi}_{sj}$ with respect to the residence time of the predecessor states of $s_j$ in path $\pi$.

$$\forall x \in C', d_{1x} \le \sum_{p=0}^{kx} t^{\pi}_{sj\text{-}p} \le d_{2x}$$

We refer to the above inequality as residence time constraint. Note that in the GTRBAC timed graph definition, the enabling condition for each event is defined with respect to the calendar clock $c_0$ of the domain, which is reset when the domain make a transition from state $s_0$ to $s_{\text{reset}}$ by traversing the edge $e_{\text{reset}}$. However, the edge $e_{\text{reset}}$ may not be included in the state path $\pi$. To ensure that a valid residence time constraint can be defined for each state in $\pi$, we concatenate a dummy path $\pi_d$: $s_{d1}.e_{d1}.s_{d2}.e_{d2}$ to the beginning of $\pi$, where $\delta(e_{d1}) = c_0$ and $\gamma(e_{d2}) = et(s_1)$. It can be easily proved that the entry time of all states in $\pi$ remains unchanged with the concatenation of path $\pi_d$. The main reason for this concatenation is that the calendar clock $c_0$ is initialized just before the first state of $\pi$ is visited, therefore, the residence time constraint can be defined for all states in $\pi$.

*Definition 9 (Traversal time of a state path)*: *The traversal time of a state path $\pi$ is defined as the sum of the residence times of all states included in $\pi$.*

Given a state path $\pi$: $s_1.e_1\ldots\ldots e_{j-1}s_j.e_js_{j+1}\ldots,e_{n-1}.s_n$, we can compute its minimal or maximal traversal time using the procedure given in Figure 5. This minimal and maximal value for state path traversal is used to determine if the given state path $\pi$ satisfies the duration and temporal constraints associated with the *component service*s as discussed in Section 4.

path-traversal-time($\pi$)
1. $\pi' \leftarrow \pi_d.\ \pi$, where $\pi_d$ is a dummy path $s_{d1}.e_{d1}.s_{d2}.e_{d2}$ with $\delta(e_{d1}) = c_0$ and $\gamma(e_{d2}) = et(s_1)$.
2. $p \leftarrow$ index(first-state($\pi'$)) and $q \leftarrow$ index(last-state($\pi'$))
3. for $i \leftarrow p$ to $q$
4. do define the residence time constraints for state $s_i$ and add it to the set of equations/inequalities for path $\pi'$.
5. Solve the system of residence time constraint generated in steps 2 and 3 for minimal or maximal value of $\sum_{i=p}^{q} t^{\pi'}_{si}$

Figure 5. Procedure for computing the minimum or maximal residence time of states in a state path.

In Section 4.2.1, we present an algorithm for verifying the correctness of a PW with respect to the GTRBAC policy of a domain. This algorithm iteratively discovers all state paths with traversal time less than

a given threshold value between a given pair of sates. To discover such paths, we need to have *a priori* information about the residence time of all the states in the corresponding domain's FSM. For this purpose, we define a minimum residence time graph (MRTG) which is generated from the GTRBAC timed graph.

*Definition 10 (Minimum Residence Time Graph)*: *A minimum residence time graph (MRTG) of a domain is a tuple MRTG = <S, E, w>, where, S and E respectively denote the sets of states and edges defined in the GTRBAC timed graph, and w is a weight function that maps each edge in the set E to a non-negative real number. For an edge $e_j$ from state $s_j$ to $s_{j+1}$, $w(e_j)$ denotes the minimum time the domain stays in GTRBAC state $s_j$ before moving to the next state $s_{j+1}$.*

For computing $w(e_j)$, we evaluate the minimum residence time of state $s_j$ over all over all possible state paths passing through edge $e_j$. Figure 6 shows a procedure for determining the weight $w$ for each edge in MRTG.

MRTG-Edge-Weights
1. Set $w(e_j) \leftarrow \infty$ for all $e_j \in E$.
2. For each pair of states $s_p, s_q \in S$ $(p \neq q)$, find a set of all simple paths $\prod_{pq}$ from $s_p$ to $s_q$.
3. For each $\pi$ in $\prod_{pq}$ compute the minimum traversal time of $\pi$. Let $t^{\pi}_{ej}$ denote the residence time of state $s_j$ such that $s_j$ is connected to its successor state $s_{j+1}$ in $\pi$ by the edge $e_j$.
   if $t^{\pi}_{ej} < w(e_j)$, then $w(e_j) \leftarrow t^{\pi}_{ej}$
4. Repeat step 3 for all state pairs $s_p, s_q \in S$

Figure 6. Procedure for determining the edge weights in a MRTG

Note that the length of any path in the minimum residence time graph defines a lower bound on the traversal time of the corresponding path in the GTRBAC timed graph. Therefore, the set all MRTG paths between state nodes $s_i$ and $s_j$ that are shorter than a given threshold value T includes all state path from $s_i$ to $s_j$ with traversal time less than T.

## 4 Composibility Verification

In this section, we describe the proposed approach for verification of workflow composibility. For verifying secure composibility of a distributed workflow, the correctness of the workflow specification needs to be evaluated against the individual as well as the collective behavior of all collaborating domains. This requirement provides a general guideline for analyzing the composibility of a given distributed workflow. In Section 4.1, we translate this requirement into a set of *workflow composibility* conditions against which the correctness of the distributed workflow is evaluated.

We use a two-step approach for verifying secure workflow composibility. In the first step, the distributed workflow specifications are analyzed for conformance with the security and access control policy of each collaborating domain. In the next step, the *cross-domain dependencies* amongst the *component services* of the workflow are verified. We use the term *cross-domain dependency* to refer to the precedence relationship between *component services* of the workflow that are provided by different domains. For instance in Figure

2, there is a cross-domain dependency between the *final estimate* preparation service provided by $ID_{CCO}$ and the *redemption payment* processing service provided by $ID_{CTO}$. For a given distributed workflow, the set $CS_{dep}$ defined in Table 5 captures all the cross-domain dependencies of the workflow.

The overall process of workflow composibility verification is depicted in Figure 1. In this process, first a projected workflow (PW) specification is generated from the distributed workflow specification for all domains. The PW specification of a domain is represented in form of a task graph as discussed in Section 2. Next a mapping is established between each task of the PW and the user-role activation required for execution of the corresponding task. After establishing the semantic mapping, a state-based representation of a PW is generated by mapping the GTRBAC based specification of the PW to all valid state paths that satisfy all the constraints included in the PW specification. The procedure for state mapping is given in Section 4.2.1. Mapping of a PW to valid state paths verifies the consistency of the distributed workflow with respect to the access control policy of the corresponding domain. However, this PW to state path mapping does not imply that the domain can satisfy the cross-domain dependency constraints amongst the *component service*s of the distributed workflow. For this purpose, all combinations of valid state paths from all domains are analyzed for satisfaction of cross-domain dependencies. In Sections 4.1.2 and 4.2.2, we discuss how the state paths from different domains are verified for preservation of cross-domain dependencies among the *component services.*

Table 5. Symbols and notations used in defining workflow composibility conditions

| Symbol/Notation | Description |
|---|---|
| $PW_i$ | Projected workflow assigned to domain $ID_i$ |
| $CS_i$ | Set of *component service*s provided by domain $ID_i$ |
| $CS_{dep}$ | Set of all cross-domain *component service*s that have a precedence relationship. $CS_{dep} = \{(c_q, c_r) \mid c_q \in CS_i, c_q \in CS_j \ (i \neq j)$, and $c_q$ precedes $c_r$ in the execution order of the distributed workflow$\}$ |
| $\pi_i$ | State path of domain $ID_i$ that satisfies WC1, WC2, and WC3 for $PW_i$ |
| $\Pi^{(i)}$ | Set of all paths that satisfies conditions WC1, WC2, and WC3 for the projected workflow $PW_i$ |
| $T_{\min}^{q,q+1}$ ($T_{\max}^{q,q+1}$) | Minimum (maximum) time between completion of intra domain *component service*s $c_q$ and $c_{q+1}$ |
| $\phi_q^{\pi i}$ ($\theta_q^{\pi i}$) | Initiation (completion) time of *component service*s $c_q$ in path $\pi_i$ |
| $[\min(\phi_q^{\pi i}), \max(\phi_q^{\pi i})]$ | Time interval during which *component service*s $c_q$ can be initiated in path $\pi_i$ |
| $[\min(\theta_q^{\pi i}), \max(\theta_q^{\pi i})]$ | Time interval during which *component service*s $c_q$ can be completed in path $\pi_i$ |
| $\Delta_i$ | Duration of the smallest calendar period that contains the enabling intervals of all role of $ID_i$ |
| $CS^{(i)}_{init}$ | Time interval during which each *component service* of domain $ID_i$ can be initiated. $CS^{(i)}_{init}=\{[\min(\phi_q^{\pi}), \max(\phi_q^{\pi})] \mid c_q \in CS_j$ and $\pi \in \Pi^{(i)}\}$ |
| $CS^{(i)}_{end}$ | Time interval during which each *component service* of domain $ID_i$ can be completed. $CS^{(i)}_{end}=\{[\min(\theta_q^{\pi}), \max(\theta_q^{\pi})] \mid c_q \in CS_j$ and $\pi \in \Pi^{(i)}\}$ |

## 4.1 Workflow Composibility Conditions

In this section, we specify the criteria for workflow composibility verification in a formal manner. In particular, we define a set of conditions against which a distributed workflow specification is evaluated. We classify these conditions as *intra-domain* and *inter-domain workflow composibility conditions*.

### 4.1.1 Intra-Domain Workflow Composibility Conditions

The intra-domain workflow composibility conditions are used to verify domain-specific projected workflow for conformance with the local GTRBAC policy of the domain. Let $TG_A$ denote the timed graph of the GTRBAC policy of domain $ID_A$, and $PW_A$ be the task graph corresponding to the projected workflow of $ID_A$. For a task $\tau_i \in PW_A$, let $\phi_i^{\pi'}$ denotes the time instant at which the processing of $\tau_i$ is initiated. We say $PW_A$ is consistent with respect to $TG_A$ if there exist a state path $\pi' = (\pi_d).(\pi) = (s_{d1}e_{d1}s_{d2}e_{d2}).(s_1e_1s_2\ldots.e_{n-1}s_n)$, such that $\delta(e_{d1}) = c_0$, $\gamma(e_{d2}) = et(s_1)$, for all $k < n$, $(s_k, s_{k+1}) \in E$ and the following conditions hold:

WC1. For each task $\tau_i \in PW_A$, there exists a sub-path, $\pi_i = s^i_1e^i_1s^i_2\ldots. e^i_{m-1}s^i_m$, of $\pi$ that satisfies the following properties

a. $index(s^i_{k+1}) = index(s^i_k) + 1$ for $0 < k < m$, where the function $index(s)$ returns the index of state $s$ in the sequence $\pi$.

b. There exists $(u, r) \in DRM(\tau_i)$ such that role $r$ is active for user $u$ in all the states of the sub-sequence $\pi_i$. As discussed in Section 3.1.1, the function $DRM(\tau_i)$ maps $\tau_i$ to a set of user role activation pair $(u', r')$ such that each $r'$ has the required permissions for processing task $\tau_i$ and $u'$ is authorized for $r'$.

c. $$t_{sd2} + \sum_{k=1}^{p-1} t_{sk} \le \phi_i^{\pi'} \le \sum_{k=1}^{q} t_{sk} - duration(\tau_i)$$

Where $p = index(s^i_1)$, $q = index(s^i_m)$, and $t_{sk}$ is the residence time of state $s_k$ in path $\pi$ and $ts_{d2}$ is the residence time of state $s_{d2}$ in $\pi'$ (see Section 3.2).

WC2. For any pair of tasks $\tau_i$ and $\tau_j$ of $PW_A$ such that $\tau_i$ precedes $\tau_j$ in the task execution order, the time associated with the completion of task $\tau_i$ is less than or equal to the time associated with the initiation of task $\tau_j$. Formally:

$$\phi_i^{\pi'} + duration(\tau_i) \le \phi_j^{\pi'},$$

WC3. For any pair of tasks $\tau_i$ and $\tau_j$ of $PW_A$ such that $\tau_i$ precedes $\tau_j$ in the task execution order with the temporal constraint requiring the delay between the completion of $\tau_i$ and initiation of $\tau_j$ to be bounded by the interval $[T_{min}, T_{max}]$, the following inequalities must hold:

$$\phi_i^{\pi'} + duration(\tau_i) \le \phi_j^{\pi'}$$

$$T_{min} \le \phi_j^{\pi'} - (\phi_i^{\pi'} + duration(\tau_i)) \le T_{max}.$$

Intuitively, the first condition (WC1) implies that for a PW to be consistent with the corresponding domain's policy, at least one state path must exist that satisfies duration constraints of each task of the PW. The workflow composibility conditions WC2 and WC3 imply that such state path must also satisfy the temporal constraints between all task pairs of the PW. These constraints include the precedence relationship and the timing constraints between task pairs of a PW as discussed in Section 2.2.

### 4.1.2 Inter-Domain Workflow Composibility Condition

The *inter-domain workflow composibility condition* defines the criterion for evaluating the correctness of workflow specification with respect to the collective behavior of all collaborating domains. In particular, this condition stipulates that the cross-domain dependencies among the *component services* in a distributed workflow specification needs to be satisfied. For this purpose, the state paths of all collaborating domains that satisfy the intra-domain workflow composibility conditions (WC1, WC2, and WC3) need to be analyzed for satisfaction of cross-domain dependencies. This analysis requires comparing the earliest and latest initiation/completion time of inter-domain *component services* or *tasks* that are involved in cross-domain dependencies. As mentioned in Section 2, a *component service* may itself be a workflow process comprising multiple tasks. The initiation time of a *component service* corresponds to the initiation time of the first task of the workflow associated with the *component service*. Similarly, the completion time of a *component service* is the completion time of the last/final task of the *component service* workflow.

With reference to the state path $\pi' = (\pi_d).(\pi) = (s_{d1}e_{d1}s_{d2}e_{d2}).(s_1e_1s_2\ldots.e_{n-1}s_n)$ considered in Section 4.1.1, The expression $t_{sd2} + \sum_{k=1}^{p-1} t_{sk} \leq \phi_i^{\pi'} \leq \sum_{k=1}^{q} t_{sk} - duration(\tau_i)$ defines a range of values for the initiation time of each task $\tau_i$ in $\pi'$ with respect to the calendar clock of the domain performing this task. We assume that the calendar clocks of all collaborating domains are synchronized at the time their policy instances are created. Note that this assumption does not restrict domains to have different periods for resetting of their calendar clocks. For instance the calendar clock of one domain may reset on a daily basis, whereas the calendar clock of another may reset on a weekly basis. With this assumption, the calendar clock values of all domains can be compared and so the cross-domain dependencies amongst the *component service*s can be verified based on the timing information provided by the domains. This timing information includes the earliest and latest time for initiation/completion of tasks that are involved in cross-domain dependencies. Figure 7 depicts the procedure for computing earliest and latest initiation times for each task of the PW. The earliest and latest completion time of any task $\tau$ can be easily computed by adding *duration*($\tau$) to the task initiation time values returned by the procedure.

task-initiation-time
INPUT: $\pi' = (\pi_d).(\pi) = (s_{d1}e_{d1}s_{d2}e_{d2}).(s_1e_1s_2\ldots e_{n-1}s_n)$, such that $\delta(e_{d1}) = c_0$, $\gamma(e_{d2}) = et(s_1)$
PW
OUTPUT: $\min(\phi_i^{\pi'})$ for each $\tau_i \in$ PW
$\max(\phi_i^{\pi'})$ for each $\tau_i \in$ PW

1. Generate a system of linear inequalities by adding:
   a. residence time constraints for each state included in $\pi'$.
   b. task initiation time constraint (composibility condition WC1) for each $\tau_i \in$ PW
   c. precedence constraint between all pairs of tasks $\tau_i, \tau_j \in$ PW such that $\tau_i$ precedes $\tau_j$ in the task execution order with the temporal constraint requiring the delay between the completion of $\tau_i$ and initiation of $\tau_j$ to be bounded by the interval $[T_{min}, T_{max}]$ (composibility conditions WC2 and WC3).
2. Solve the system of constraints generated in step 1 with the objective of minimizing $\Sigma_i\phi_i$. The value assigned to each $\phi_i$ equals $\min(\phi_i^{\pi'})$.
3. Solve the system of constraints generated in step 1 with the objective of maximizing $\Sigma_i\phi_i$. The value assigned to each $\phi_i$ equals $\max(\phi_i^{\pi'})$.

Figure 7. Procedure for computing the earliest and latest initiation time of each task in a projected workflow

Proposition 1: Given a task $\tau_i \in$ PW, the time $\min(\phi_i^{\pi'})$ ($\max(\phi_i^{\pi'})$) computed using the task initiation time procedure is the earliest (latest) time at which the task $\tau_i$ can be initiated in state path $\pi'$ that satisfies intra-domain workflow composibility conditions WC1, WC2, and WC3 for PW.

*Proof of this proposition is provided in the Appendix.*

The cross-domain dependencies amongst the *component service*s can be represented in an algebraic form based on the task initiation and completion information provided by collaborating domains. The notations and symbols used in this representation of cross-domain dependencies are listed in Table 5.

For any pair of cross-domain *component service*s $c_q$ and $c_r$ such that $(c_q, c_r) \in CS_{dep}$, the following set of algebraic constraints captures both the intra-domain and cross-domain dependency constraints between $c_q$ and $c_r$.

$$v\Delta_i + \theta_q^{\pi i} \le w\Delta_j + \phi_y^{\pi j} \qquad (v, w \in Z^+) \qquad \text{(I)}$$

$$\min(\theta_q^{\pi i}) \le \theta_q^{\pi i} \le \max(\theta_q^{\pi i}) \qquad \text{(II)}$$

$$\min(\phi_q^{\pi i}) \le \phi_q^{\pi i} \le \max(\phi_q^{\pi i}) \qquad \text{(III)}$$

$$\min(\theta_r^{\pi j}) \le \theta_r^{\pi j} \le \max(\theta_r^{\pi j}) \qquad \text{(IV)}$$

$$\min(\phi_r^{\pi j}) \le \phi_r^{\pi j} \le \max(\phi_r^{\pi j}) \qquad \text{(V)}$$

$$T_{\min}^{q,q+1} \le \phi_q^{\pi i} - \theta_q^{\pi} i \le T_{\max}^{q,q+1} \qquad \text{(VI)}$$

$$T_{\min}^{r-1,r} \le \phi_r^{\pi j} - \theta_r^{\pi j} \le T_{\max}^{r-1,r} \qquad \text{(VIII)}$$

Constraint (I) implies that the *component service* $c_q$ must be completed before $c_r$ is initiated in any calendar period. The variables $\theta_q^{\pi i}$and $\phi_r^{\pi j}$ denote the completion and initiation times of *component service*s $c_q$ and $c_r$ in state paths $\pi_i$ and $\pi_j$ respectively. The bounds on these two variables are specified in constraints (II) - (V) given above. Constraints (VI) and (VII) specify timing constraints between the intra-domain

*component service*s. These timing constraints are computed while generating PW specification as discussed in Section 2.

If the solution set of the above system of inequalities generated for all $(c_q , c_r) \in CS_{dep}$ is non-empty then the state paths $\pi_i$ and $\pi_j$ jointly satisfy all the cross-domain dependencies between $PW_i$ and $PW_j$. Based on this implication, the following condition for verifying the workflow composibility with respect to temporal dependencies among the *component service*s can be defined.

WC4. Two state paths $\pi_i$ and $\pi_j$, respectively satisfying conditions WC1, WC2, and WC3 for the projected workflows assigned to $ID_i$ and $ID_j$, are consistent if they satisfy all the cross-domain dependencies included in the set $CS_{dep}$.

### 4.1.3 Overall Criteria for Workflow Composibility

Based on the *intra-domain* and *inter-domain workflow composibility* conditions, we provide the following overall criteria for workflow composibility.

Given a distributed workflow S, a set of S's projected workflows $PW = \{PW_1,\ldots,PW_n\}$, a set of cross-domain dependencies among *component service*s $CS_{dep} = \{(c_i^q , c_j^r) |\ c_i^q$ precedes $c_j^r$ and $1 \leq i,j \leq n$ and $i \neq j\}$, and a set F of FSMs, modeling domains' GTRBAC policies. Let $\Pi^{(i)}$ denote the set of state paths of $ID_i$ such that each path in $\Pi^{(i)}$ satisfies workflow composibility conditions WC1, WC2, and WC3 for $PW_i$. We say that S is composable with respect to F if the following hold:

1. For any $ID_i$, the set of paths $\Pi^{(i)}$ is non-empty.
2. There exists a tuple $(\pi_1, \pi_2, \ldots,\pi_n) \in \Pi^{(1)} \times \Pi^{(2)} \times \ldots \times \Pi^{(n)}$ such that $(\pi_1, \pi_2, \ldots,\pi_n)$ satisfy all cross-domain dependencies $(c_i^q , c_j^r) \in CS_{dep}$, where $1 \leq i,j \leq n$.

*Example* 3: Consider the projected workflows $PW_{CTO}$ and $PW_{CCO}$ assigned to $ID_{CTO}$ and $ID_{CCO}$ as shown in Figure 2. The GTRBAC policies of these domains are listed in Table 3 and the corresponding FSMs ($F_{CTO}$ and $F_{CCO}$) are shown in Figure 4. For $ID_{CTO}$, we consider the state transition path $\pi_1 = s_3.e_3.s_4.e_4.s_5.e_5.s_6.e_6.s_7.e_7.s_6$ of $F_{CTO}$ that satisfies the composibility conditions WC1, WC2, and WC3 for $PW_{CTO}$. In the path $\pi_1$, all the states $s_3$, $s_4$, $s_5$, $s_6$, and $s_7$ support execution of *exemption processing* (EP) task that requires activation of the role TEP by an authorized user (in this case $u_1$). The task of *payment processing* (PP) can be performed in states $s_6$ or $s_7$ in which the role TPP is active for user $u_2$. Finally, in state $s_7$ the *refund adjustment* (RA) *task* can be processed by $u_3$ assuming the role TRP. For the projected workflow $PW_{CCO}$, the state path $\pi_2 = s_{21}.e_{26}.s_{22}.e_{27}.s_{23}.e_{28}.s_{21}$ of $F_{CCO}$ satisfies the composibility conditions WC1, WC2, and WC3. In the path $\pi_2$, the *initial assessment* (IA) task can be processed in all states included in $\pi_2$. The tasks of preparing *final estimate* (FE) and *clearance processing* (CP) can only be performed in

state $s_{23}$. The constraints on the initiation and completion times of the *component service*s imposed by state paths $\pi_1$ and $\pi_2$ are listed in Table 6(a) and Table 6(b).

Table 6(a). Constraints imposed by $\pi_1$ on the initiation and completion times of *component services* of $ID_{CTO}$

| $\Delta_{CTO}$ | 1440 |
|---|---|
| $[\min(\phi_{EP}^{\pi 1}), \min(\phi_{EP}^{\pi 1})]$ | [540, 740] |
| $[\min(\phi_{RA}^{\pi 1}), \min(\phi_{RA}^{\pi 1})]$ | [620, 800] |
| $[\min(\theta_{EP}^{\pi 1}), \min(\theta_{EP}^{\pi 1})]$ | [560, 760] |
| $[\min(\theta_{PP}^{\pi 1}), \min(\theta_{PP}^{\pi 1})]$ | [680, 840] |
| $\phi_{RA}^{\pi 1} - \theta_{EP}^{\pi 1}$ | ≥ 60 and ≤235 |

Table 6(b). Constraints imposed by $\pi_2$ on the initiation and completion times of *component services* of $ID_{CCO}$

| $\Delta_{CCO}$ | 1440 |
|---|---|
| $[\min(\phi_{IA}^{\pi 2}), \min(\phi_{IA}^{\pi 2})]$ | [480, 585] |
| $[\min(\phi_{FE}^{\pi 2}), \min(\phi_{FE}^{\pi 2})]$ | [505, 610] |
| $[\min(\phi_{CP}^{\pi 2}), \min(\phi_{CP}^{\pi 2})]$ | [605, 710] |
| $[\min(\theta_{IA}^{\pi 2}), \min(\theta_{IA}^{\pi 2})]$ | [485, 590] |
| $[\min(\theta_{FE}^{\pi 2}), \min(\theta_{FE}^{\pi 2})]$ | [565, 670] |
| $[\min(\theta_{CP}^{\pi 2}), \min(\theta_{CP}^{\pi 2})]$ | [615, 720] |
| $\phi_{FE}^{\pi 2} - \theta_{IA}^{\pi 2}$ | ≥ 20 and ≤55 |
| $\phi_{CP}^{\pi 2} - \theta_{FE}^{\pi 2}$ | ≥ 40 and ≤170 |

To verify whether the state paths $\pi_1$ and $\pi_2$ satisfy all the cross-domain dependencies between $ID_{CTO}$ and $ID_{CCO}$, the following system of constraints is generated and solved for a feasible solution.

(a1) $v\Delta_{CCO} + \theta_{IA}^{\pi 2} \le w\Delta_{CTO} + \phi_{EP}^{\pi 1}$; (a2) $w\Delta_{CTO} + \theta_{EP}^{\pi 1} \le v\Delta_{CCO} + \phi_{FE}^{\pi 2}$;
(a3) $v\Delta_{CCO} + \theta_{FE}^{\pi 2} \le w\Delta_{CTO} + \phi_{RA}^{\pi 1}$; (a4) $v\Delta_{CCO} + \theta_{PP}^{\pi 1} \le w\Delta_{CCO} + \phi_{CP}^{\pi 2}$;
(a5) $480 \le \phi_{IA}^{\pi 2} \le 585$; (a6) $505 \le \phi_{FE}^{\pi 2} \le 610$; (a7) $605 \le \phi_{CP}^{\pi 2} \le 710$;
(a8) $485 \le \theta_{IA}^{\pi 2} \le 590$; (a9) $565 \le \theta_{FE}^{\pi 2} \le 670$; (a10) $615 \le \theta_{CP}^{\pi 2} \le 720$;
(a11) $540 \le \phi_{EP}^{\pi 1} \le 740$; (a12) $620 \le \phi_{RA}^{\pi 1} \le 800$; (a13) $560 \le \theta_{EP}^{\pi 1} \le 760$; (a14) $680 \le \theta_{PP}^{\pi 1} \le 840$;
(a15) $20 \le \phi_{FE}^{\pi 2} - \theta_{IA}^{\pi 2} \le 55$; (a16) $40 \le \phi_{CP}^{\pi 2} - \theta_{FE}^{\pi 2} \le 170$; (a17) $60 \le \phi_{RA}^{\pi 1} - \theta_{EP}^{\pi 1} \le 235$;
(a18) $v \ge 0$ and integer; $v \ge 0$ and integer

One of the feasible solutions to the above system of inequalities have the following assignment: $v = w = 1$, $\phi_{IA}^{\pi 2} = 580$, $\theta_{IA}^{\pi 2} = 590$, $\phi_{EP}^{\pi 1} = 590$, $\theta_{EP}^{\pi 1} = 610$, $\phi_{FE}^{\pi 2} = 610$, $\theta_{FE}^{\pi 2} = 670$, $\phi_{RA}^{\pi 1} = 670$, $\theta_{PP}^{\pi 1} = 710$, $\phi_{CP}^{\pi 2} = 710$, $\theta_{CP}^{\pi 2} = 720$. With this assignment the initiation time of the *tax redemption workflow* of Figure 2 is 580 minutes with respect to the calendar clock of $ID_{CCO}$, i.e., the *tax redemption workflow* can be initiated at 9:40 am. Note that the above system of inequalities is satisfied if we add the term *km*' to both sides of constraints a1 - a4, listed above. Where *m'* is the *lowest common multiple* of $v\Delta_{CTO}$ and $w\Delta_{CCO}$, and *k* is any non-negative integer. For the above assignment, *m'* equals 1440. With reference to the FSM of Figure 4, *m'* corresponds to a periodic time instant at which $ID_{CTO}$ returns to state $s_3$ (first state of $\pi_1$) and $ID_{CCO}$ returns to state $s_{21}$ (first state of $\pi_2$). This means that the *tax redemption workflow* can be repeatedly initiated after every 1440 minutes (24 hours) since its previous initiation. In other words, the workflow can be initiated every day at 9:40 am.

### 4.2 Composibility Verification Algorithm

In this section, we present two algorithms for verifying the composibility of a given distributed workflow. The first algorithm, presented in Section 4.2.1, verifies the correctness of a domain specific

projected workflow by finding all valid state paths that satisfy the intra-domain workflow composibility conditions. We use the term valid state path to refer to a path that satisfies composibility conditions WC1, WC2, and WC3 for the given projected workflow. The second algorithm, presented in Section 4.2.2, analyzes all inter-domain path combinations for satisfaction of the inter-domain workflow composibility condition.

### 4.2.1 PW Consistency Verification

For discovering all valid state paths for a given PW, we use functions *smap* and *emap* iteratively. The function *smap* maps a given task of a PW to a set of GTRBAC states that have the appropriate user-role activation required for task execution. We refer to such states as the entry states of the task. Formally: $smap(\tau) = \{s \mid s \in S \text{ and } \exists (u, r) \in DRM(\tau) \text{ such that role } r \text{ is active for } u \text{ in state } s\}$. The function *emap* maps an edge $(\tau_i, \tau_j)$, representing successive tasks in a PW graph, to a set of state paths $\Pi_{ij}$ that satisfy the composibility conditions WC1, WC2, and WC3 for tasks $\tau_i$ and $\tau_j$.

```
WPS
INPUT: PW (project workflow graph of a domain)
       TG (GTRBAC timed graph of domain)
       MRTG (Minimum residence time graph of a domain)
OUTPUT: Π1n (set of all state paths that satisfy all the intra-
domain composibility conditions for PW)
1. for τi ∈ PW
2.   do color[τi] ← white
3.    for each τj ∈ PW
4.      do Πij ← ϕ
5. color[τ1] ← dark gray
6. Q ← τ1
7. while Q ≠ ϕ
8.   do τi ← dequeue(Q)
9.      for each τj such that (τi, τj) ∈ E[PW]
10.    do if color[τj] = white  or color[τj] = light gray
11.          then Πij ←  edge-mapping(TG, MRTG, τi, τj)
12.               if τi ≠ τ1
13.                  then if color[τj] = light gray
14.                            then Π1j←path-extend(Π1i, Πij) ∩ Π1j
15.                            else  Π1j←path-extend(Π1i, Πij)
16.                  if there exists τk such that (τk, τj) ∈ E[PW]
17.                     and τk≠τi and color[τk] ≠ black
18.                     then color[τj] ← light gray
19.                     else color[τj] ← gray
20.                           Q ← Q ∪ τj
21.       color[τi] ← black
```

```
edge-mapping(TG, MRTG, τi, τj)
    1.  for each p ∈ smap(τi) and q ∈ smap(τj)
    2.    do Π ← find-all-paths(p, q, MRTG)
    3.       for each π' ∈ Π
    4.          do if π' does not satisfy any of the
                composibility conditions WC1, WC2, and
                WC3 for τi and τj
    5.                then Π← Π - π'
    6.  return Π

path-extend(PW, Π1i, Πij)
1.  Π'1j ← ϕ
2.  for each π', π such that π'∈ Π1i and π∈ Πij
3.    do πi ← maximal sub-path of  both π' and π
4.         such that finish(πi, π') and start(πi, π)
5.         if πi = π then π1j ← π'
6.                   else π1j  ← concat(π'/πi, π)
7.         if π1j satisfies the conditions WC1, WC2,
    and WC3 for all task pairs τp, τq such that (τp,
    τq) ∈ E[PW] and color[τp] ≠ white and (τq =τj
    or color[τq] ≠ white)
8.  then Π' 1j←Π' 1j∪ π1j
9.  return Π1j
```

Figure 8. Algorithm for discovering the valid state paths of a given PW.

Figure 8 shows the pseudo-code for the edge-mapping procedure. This procedure first discovers all state paths with traversal time less than a threshold value between all entry states of tasks $\tau_i$ and $\tau_j$. The threshold value corresponds to the maximum time allowed for completion of tasks $\tau_i$ and $\tau_j$. The discovered paths are

then analyzed for satisfaction of composibility conditions WC1, WC2, and WC3 for tasks pair ( $\tau_i$ , $\tau_j$ ). We use the minimum residence time graph (MRTG) defined in Section 3.4 to discover these state paths. As mentioned in Section 3.4, the set all MRTG paths between state nodes $s_1$ and $s_2$ that are shorter than a given threshold value T includes all state paths from $s_1$ to $s_2$ with traversal time less than T.

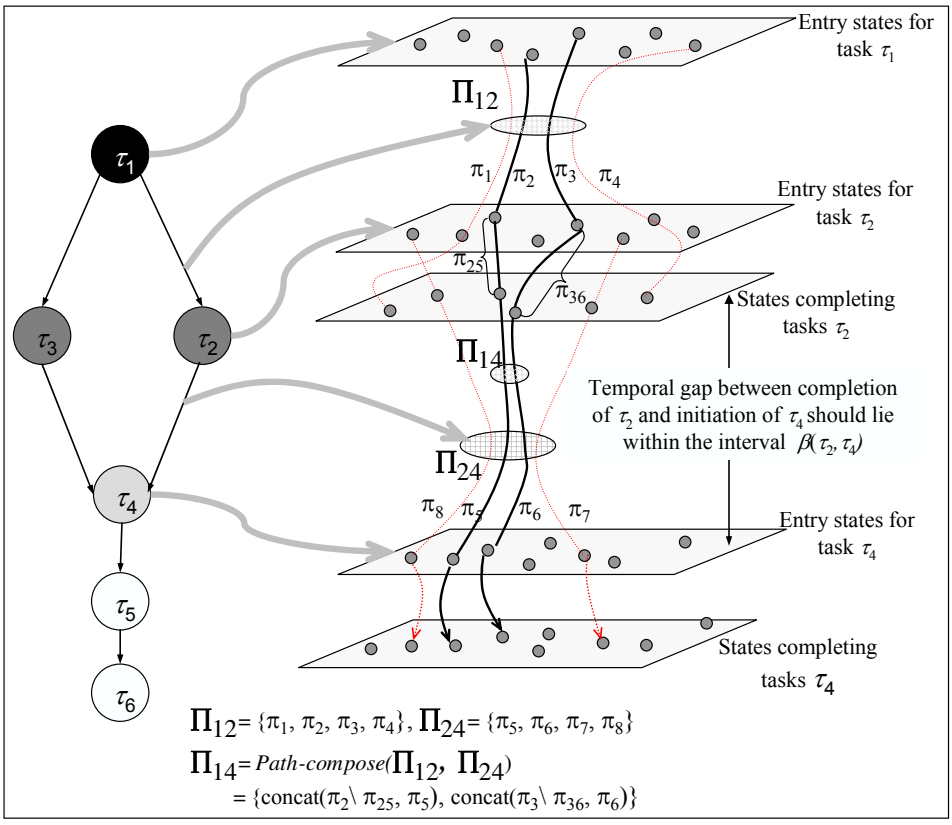
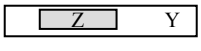
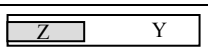
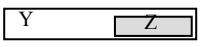



Figure 9. Mapping between PW graph and state paths of a domain's FSM

Table 6. Path relations

| Let X = <$x_0$=($s_0$,$e_0$), ….. $x_{m-1}$=($s_{m-1}$,$e_{m-1}$)>, Y = <$y_0$=($s_0$',$e_0$'),… .. $y_{n-1}$=($s_{n-1}$',$e_{n-1}$') >, and Z = <$z_0$=($s_0$",$e_0$"),…., $z_{k-11}$=($s_{k-1}$",$e_{k-1}$") > be state paths. | | |
|---|---|---|
| Relation Predicate | Definition | Illustration |
| *during*(Z, Y) | There exists an index *i* of Y, such that for all *j* = 0, 1, ..., *k*-1, $z_j = y_{i+j}$. |   |
| *Start*(Z, Y) | For all *j* = 0, 1, ..., *k*-1, $z_j = y_j$ |   |
| *finish*(Z, Y) | For all *j* = 0, 1, ..., *k*-1, $z_j = y_{(n-1)-(k-1)+j}$, where $y_{(n-1)}$ is the last element of the state transition path Y. |   |
| *Meet*(Z, Y, X) | There exists an index *p* of path X, such that for all *i* = 0, 1, ..., *n*-1, and for all *j* = 0, 1, ..., *k*-1, $y_i = x_{p+i}$, $z_j = x_{p+n+j}$. |   |

The complete state path of a PW can be composed by incrementally extending the state paths corresponding to successive edge mappings as shown in Figure 9. In this figure, $\pi_2 \in \Pi_{12}$ represents a state path from the initiation of task $\tau_1$ to the completion of task $\tau_2$, and $\pi_5 \in \Pi_{24}$ is a state path from the initiation

of task $\tau_2$ to the completion of task $\tau_4$. These two state paths overlap for the execution duration of task $\tau_2$ and therefore can be combined to compose a state path from $\tau_1$ to $\tau_4$. Let $\pi_{25}$ be the overlapping sub-path of $\pi_2$ and $\pi_5$ such that the following path relations, defined in Table 6, hold: *finish*($\pi_{25}$, $\pi_2$) and *start*($\pi_{25}$, $\pi_5$). Moreover, for all states $s$ in $\pi_{25}$, $s \in smap(\tau_2)$. The state path $\pi_2$ can be written in a concatenated form as $\pi_2 = \pi'.\pi_{25}$, where $\pi'$ is a sub-path of $\pi_2$ such that *meet*($\pi'$, $\pi_{25}$, $\pi_2$ ) is true. A state path $\pi_{14}$ can be composed by concatenating $\pi_5$ to the end of $\pi'$. The path $\pi_{14}$ need to be checked for satisfaction of composibility conditions WC1, WC2, and WC3 for both task pairs ($\tau_1$ , $\tau_2$) and ($\tau_2$ , $\tau_4$).

Figure 8 shows an algorithm *workflow path search* (WPS) for discovering all valid state paths for a given PW. The algorithm takes input the task graph of a PW, the FSM of a domain's GTRBAC policy represented as timed graph TG, and the minimum residence time graph of TG. During the path search, the PW graph is traversed in a breadth first manner to explore all the state paths from the source node $\tau_1$ to all other nodes of the PW that satisfy the composibility conditions WC1, WC2, and WC3 for all successive pairs of tasks in the PW. The path set $\prod$ is indexed by the indices of source and destination nodes of the PW graph. For a given source $\tau_1 \in$ PW and destination $\tau_i \in$ PW, $\prod_{1i}$ denotes the set of all valid state paths that satisfy all the temporal ordering and duration constraints for successful completion of a PW with $\tau_1$ as a source node and $\tau_i$ as a terminal node in the PW graph.

To keep track of the tasks whose state paths from the source node have been discovered, the algorithm *WPS* colors each vertex in the PW graph as white, dark gray, light gray, or black. All task vertices in the PW graph start out as white. A task vertex $\tau_i$ becomes dark gray after the discovery of all the valid state paths from the source vertex $\tau_1$ to $\tau_i$. Incase a task has multiple adjacent vertices that precede $\tau_i$ in the execution order, $\tau_i$ becomes light gray after its first discovery and remains light gray until the consistency of the path set $\prod_{1i}$ has been verified for all the adjacent vertices preceding $\tau_i$ in the workflow execution order. After this consistency verification, vertex $\tau_i$ is colored from light gray to dark gray. A dark gray vertex $\tau_i$ becomes black after all the valid state paths from $\tau_1$ to all the adjacent vertices of $\tau_i$ have been discovered. The algorithm terminates when all the vertices of the PW have been colored black. At this point all the valid state paths from $\tau_1$ to the final task have been discovered.

Correctness of the algorithm: To verify that a PW conforms to the GTRBAC policy of the designated domain, we need to find at least one state path that satisfies all the intra-domain workflow composibility conditions WC1, WC2, and WC3 described in Section 4.1. The set of all state paths returned by the *WPS* procedure meet this requirement for PW verification. It can also be noted that the set of state paths returned by *WPS* are exhaustive, i.e., if any path satisfies the intra-domain composibility conditions for a given PW, then it is included in the set of paths discovered by *WPS*.

Theorem 1: Let $G_X$ be a graph representing the specification of a PW assigned to domain $ID_X$. Suppose $\tau_1$ is a distinguished source vertex of $G_X$ that initiates the PW. Let $\tau_j$ ($\neq \tau_1$) be any task vertex in $G_X$. In the *WPS* procedure, after the task vertex $\tau_j$ is colored dark gray, following properties hold for each state transition path $\pi$ in the path set $\prod_{1j}$.

1. Composibility condition WC1 is satisfied for task $\tau_j$ and all tasks $\tau_i$ preceding $\tau_j$ in task execution order.
2. Composibility conditions WC2 and WC3 are satisfied for all task pairs ($\tau_i$, $\tau_j$) such that ($\tau_i$, $\tau_j$) $\in$ E[$G_X$]. In addition, WC2 and WC3 are also satisfied for all task pairs ($\tau_p$, $\tau_q$) such that ($\tau_p$, $\tau_q$) $\in$ E[$G_X$] and $\tau_p$, $\tau_q$ precedes $\tau_j$ in the task execution order.

*Proof given in Appendix.*

### 4.2.2 Cross-Domain Dependency Verification Algorithm

In this section, we present a simple algorithm, shown in Figure10, for verification of distributed workflow with respect to cross-domain dependencies. The symbols and notations used in this procedure are described in Table 5. The cross-domain dependency verification is performed by a central site. Given $\Pi^{(i)}$, $CS^{(i)}_{init}$, and $CS^{(i)}_{end}$ ($1 \leq i \leq n$) for all collaborating domains, and the set $CS_{dep}$, the algorithm analyzes all cross-domain path combinations for satisfaction of the precedence relationship between *component service*s specified in $CS_{dep}$. In this analysis, a system of inequalities defining precedence relationship among the *component service*s is generated and solved for each *n*-ary tuple $y = (\pi_1, \pi_2, \ldots, \pi_n)$. A feasible solution to this system of inequalities implies the following:

- The state path combination ($\pi_1$, $\pi_2$, …,$\pi_n$) corresponding to the tuple *y*, satisfies all the cross-domain dependency relationships specified in the distributed workflow specification.
- For the above path combination, the projected workflow in $ID_i$ can be supported at any time included in the solution space of the system of inequalities generated for *y*.

If no feasible solution exists for any tuple $y \in (\Pi^{(1)} \times \Pi^{(2)} \times \ldots \times \Pi^{(n)})$, then the verification procedure returns No. In this case, the given distributed workflow cannot be supported. This is stated in the following theorem.

Theorem 2: Given $\Pi^{(i)}$, $CS^{(i)}_{init}$, and $CS^{(i)}_{end}$ for each domain $ID_i$ ($1 \leq i \leq n$) and the set $CS_{dep}$, if the cross domain verification procedure fails, then the cross domain dependencies in the given set $CS_{dep}$ cannot be satisfied, Accordingly, the corresponding distributed workflow cannot be supported.

*Proof of this theorem is given in Appendix.*

Comment on Complexity: The proposed workflow composibility verification approach has a high computational complexity as observed in many other similar problems [Lin88, Wes89]. This high complexity is mainly due to the exhaustive state path searches performed in the projected workflow verification step. In this step, all state paths of length less than a given threshold value are discovered

between the entry states of successive tasks of the projected workflow. The problem of finding all length limited paths between any two nodes in a graph is at least as difficult as solving the S-T PATH problem, which is defined as finding all simple paths from a node *s* to another node *t* in a graph. The S-T PATH problem is proved to be #P-Complete [Val79].

The complexity of the composibility verification problem discussed in this paper can be significantly reduced, if instead of discovering all valid state paths, only *m*-shortest paths are discovered, where $m > 1$. However, this heuristic will consider a sub-set of all valid state paths for workflow verification and in the worst case may declare a correct workflow specification as incomposable.

```
cross-domain-dependency-verification
INPUT: Π^(i) for all IDs (1≤ i ≤ n)
       CS^(i)_init for all IDs (1≤ i ≤ n)
       CS^(i)_end for all IDs (1≤ i ≤ n)
       CS_dep
OUTPUT: {Yes, No}
1.  Y ← Π^(1) × Π^(2) × …× Π^(n)
2.  for each y ∈ Y
3.      do for each ((c_q, c_r) ∈ CS_dep define cross-domain dependency constraints between c_q
4.              and c_r end for /*end inner for loop of line 3 */
5.      if the solution space to the system of inequalities generated for y ∈ Y is non empty
6.          then return Yes
7.     end for /*end outer for loop of line 2 */
8.   return No
```

Figure 10. Algorithm for verifying distributed workflow with respect to cross-domain dependencies among *component services*.

## 5 Related Work

Both workflow composibility and scheduling verification of distributed workflow*s* are daunting issues in their own respect and have been addressed in the literature as independent topics [Den03, Ber99a, Ada98, Cao04, Esh04, Gon96, He05]. For example, Deng et. al. [Den03] have proposed an architectural approach for secure composition of software components that ensures satisfaction of system-wide security constraints. In this approach, security analysis of the composed system is driven by incremental propagation of global security constraints onto individual software components. This approach is particularly useful in federated systems in which system wide security constraints can be enforced by configuring individual components according to the global security policies. However, the approach is limited to non-autonomous and reentrant system behavior.

One of the most effective and widely-used approach for automatic verification of component-based systems is model checking. In model checking, the overall system behavior is represented in form of a state-transition graph model and is compared with temporal logic formulae modeling the intended or undesired properties of the component-based system, services, or workflows. Various temporal logics have been proposed for model checking. These include linear time logic (LTL), proposition dynamic logic (PDL), and

computation time logic (CTL) [Cla86, Har00, Gab70]. Based on these temporal logics, many model checking based approaches have been proposed in literature for verification of e-services [Ber03, Ber05, Bon05, Fu02, Fos03]. However, the temporal logics CTL, LTL, and PDL do not support specification of quantitative temporal constraints and cannot be used for verification of systems or workflows with real-time constraints.

Braberman *et. al.* [Bra05] have proposed an automata theoretic approach to check if a given scenario with quantitative temporal constraints can be supported by a system. They use an existential semantics to find a system trace that satisfies the scenario. To use this approach for workflow composibility verification, a global meta-policy that captures the collective behavior of all domains is needed. However, as discussed in the Introduction, such meta-policy cannot be created for an environment comprising autonomous and non-reentrant domains.

## 6 Conclusion

In this paper, we have proposed an approach for verifying the secure composibility of distributed workflows in a collaborative environment comprising autonomous domains. The objective of workflow composibility verification is to ensure that all the users or processes executing the designated workflow tasks have proper authorization and their activities within the context of workflow specification cannot cause security breach in any domain. The proposed approach achieves this objective by verifying the distributed workflow specifications against the access control policies of all domains collaborating for workflow execution. A key challenge in this verification process is posed by the time-dependent access control policies of collaborating domains which are specified using GTRBAC model. The GTRBAC policy of a domain contributes to its non-reentrant behavior which is modeled as a time augmented FSM. The proposed approach verifies workflow composibility by exploring the FSM of each domain to find state paths that satisfy the given workflow specifications. This workflow composibility verification is performed without creating a unified global FSM which is required for model checking-based approaches for composibility verification. Therefore, the proposed approach is unique and does not compromise the autonomy and privacy of collaborating domains.

We have discussed the proposed approach in the context of verifying workflow specifications for conformance with the security and access control policies of domains. However, the proposed approach is generic and can be broadly applied to many distributed applications requiring interactions among non-reentrant and autonomous systems.

## 7 References

[Ada98] N. R. Adam, V. Athluri, W. Huang. "Modeling and Analysis of Workflows Using Petri Nets." *Journal of Intelligent Information Systems*, 10, pp. 131-158, 1998.

[Agr82] J. R. Agre, S. K. Tripathi, "Modeling Reentrant and Nonreentrant Software." *Proc. ACM SIGMETRICS conference on Measurement and modeling of computer systems*, 12(3), pp.163-178, 1982.

[Alu93] R. Alur, C. Courcoubetis, and D. Dill, "Model Checking in Dense Real-Time," *Information and Computation*, Vol. 104, No. 1, 1993, pp. 2-34.

[Alu94] R. Alur and D. Dill, "A Theory of Timed Automata," Theoretical Computer Science, Vol. 126, No. 2, 1994, pp. 183-235.

[Ant01] A. I. Antón and J. B. Earp. "Strategies for Developing Policies and Requirements for Secure E-Commerce Systems." *Recent Advances in E-Commerce Security and Privacy, Kluwer Academic Publishers*, pp. 29-46, 2001.

[Bel02] A. Belokosztolszki and K. Moody, "Meta-Policies for Distributed Role-Based Access Control Systems," *proc. of the 3rd IEEE International Workshop on Policies for Distributed Systems and Networks*, 2002.

[Ber99] E. Bertino, E. Ferrari and V. Atluri. "The Specification and Enforcement of Authorization Constraints in Workflow Management Systems." *ACM Transactions on Information and System Security*, Vol.2, No. 1, pp. 65-104. 1999.

[Ber99b] E. Bertino, C. Bettini, E. Ferrari, and P. Samarati, "An Access Control Model Supporting Periodicity Constraints and Temporal Reasoning," *ACM Transactions on Database Systems*, Vol. 23, No. 3, pp. 231-285.

[Ber03] D. Bedrardi, D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Mecella, "Automatic Composition of E-Services that Export Their Behavior," *Proc. of International Symposium on Service Oriented Computing*, 2003, pp. 43-58.

[Ber05] D. Bedrardi, D. Calvanese, G. D. Giacomo, R. Hull, and M. Mecella, "Automatic Composition of Transition Based Semantic Web services with Messaging," *proc. of the the 31st VLDB Conference, Trondheim*, Norway, 2005, pp. 613-623.

[Bon05] Y. Bontemps, P. Heymans, and P-Y Schobbens, "From Live Sequence Charts to State Machines and Back: A guided Tour," *IEEE Transactions on Software Engineering*, Vol. 31, No. 12, pp. 999-1013.

[Bra05] V. Braberman, N. Kicilof, and A. Olivero, "A Scenario-Matching Approach to the Description and Model Checking of Real-Time Properties," *IEEE Transactions on Software Engineering*, Vol. 31, No. 12, pp. 1028-1041.

[Cao04] J. Cao, S.A.Jarvis, S. Saini and G. R. Nudd, "GridFlow: Workflow Management for Grid Computing," *proc. 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid03)*, pp.198 – 205. 2003.

[Car88] C. Canright, "Will Real-Time Systems Spell the End for Batch Processing," *Bank Admin.,* Vol. 64, No. 9, September 1988, pp. 42-46.

[Coh02] E. Cohen, R. K. Thomas, W. Winsborough and D. Shands, "Models for Coalition-Based Access Control (CBAC)," *proc. of the seventh ACM symposium on Access control models and technologies*, pp. 97-106, 2002.

[Cla86] E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic Verification of Concurrent Systems Using Temporal Logic Specifications," *ACM Transactions on Programming languages and Systems*," Vol. 8, No. 2, pp. 244-263.

[Den03]Y. Deng, J. Wang, J.J.P Tsai and K. Beznosov, "An Approach for Modeling and Analysis of Security System Architectures," *IEEE Transactions on Knowledge and Data Engineering,* 15(5), pp.1099 - 1119, 2003.

[Erm80] L. D. Erman, F. H.-Roth, V. R. Lesser, and D. R. Reddy, "The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty," *ACM Computing Surveys*, June 1980, pp. 213-252.

[Esh04] R. Eshuis, R. Wieringa, "Tool Support for Verifying UML Activity Diagrams," *IEEE Transactions on Software Engineering*, 30(7), pp. 437 – 447, 2004.

[Fos03] H. Foster, S. Uchitel, J. Magee, and J. Kramer, "Model-based Verification of Web Service Compositions," *proc. of the 18th IEEE International Conference on Automated Software Engineering*, 2003.

[Fu02] X. Fu, T. Bultan, and J. Su, "Formal Verification of e-Services and Workflows," *proc. of Workshop on Web Services, E-Business, and the Semantic Web*, May 2002, pp. 188-202.

[Gab80] D. Gabbay, A. Pnuelli, S. Shelah, and J. Stavi, "On the Temporal Analysis of Fairness," *proc. of Seventh ACM Symposium on Principles of Programming Languages*, 1980, pp. 163-173.

[Gac98] C. Gacek, "Detecting Architectural Mismatches during Systems Composition," PhD. Thesis, University of Southern California, 1998.

[Gho93] S. Ghosh, "NOVADIB: a Novel Architecture for Asynchronous, Distributed, Real-Time Banking Modeled on Loosely Coupled Parallel Processors," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 23, No. 3, May 1993, pp. 917-927.

[Gon96] L. Gong and X. Qian, "Computational Issues in Secure Interoperation," *IEEE Transaction on Software and Engineering*, 22(1), pp.43-52, 1996.

[Har00] D. Harel, D. Kozen, and J. Tiuryn. Dynamic Logic. The MIT Press, 2000.

[He05] Q. He and A.I. Antón, "Deriving Access Control Policies from Requirements Specifications and Database Designs," *NCSU CSC Technical Report #TR-2004-24*, 2004.

[Jaf91] M. S. Jaffe, N. G. Leveson, M. P. E. Heimdahl, and B. E. Melhart, "Software Requirements Analysis for Real-Time Process-Control Systems," *IEEE Transactions on Software Engineering*, Vo. 17, No. 3, pp. 241-258.

[Jos03] J. Joshi. A Generalized Temporal Role Based Access Model For Developing Secure Systems. *PhD Thesis*. Purdue University, West Lafayette, IN. 2003

[Jos05a] J. Joshi, E. Bertino, U. Latif and A. Ghafoor. A Generalized Temporal Role Based Access Control Model for Developing Secure Systems. *IEEE Transactions on Knowledge and Data Engineering,* 17(1), pp. 4-23, 2005.

[Jos05b] J. Joshi, E. Bertino, and A. Ghafoor, "Analysis of Expressiveness and Design Issues for a Temporal Role Based Access Control Model," *IEEE Transactions on Dependable and Secure Computing*, Vol. 2, No. 2, pp. 157-175.

[Jun04] J. Jung, W. Hur, S. Kang and H. Kim, "Business Process Choreography for B2B Collaboration," *IEEE Internet Computing*, 8(1), pp. 37-45, 2004.

[ITU96] ITU. Message Sequence Charts. *Recommendation Z.120, International Telecomm. Union, Telecomm. Standardization Sector*, 1996.

[Kru04] I. Krüger, "Service Specification with MSCs and Roles," *proc. IASTED International Conference on Software Engineering*, Innsbruck, 2004.

[Lam00] B. W. Lampson, "Computer Security in the Real World," *Annual Computer Security Applications Conference*, December 11-15, 2000.

[Lan01] X. E. Landwehr, "Computer Security," International Journal of Information Security, Vol. 1, No. 1, August 2001, pp. 3 – 13.

[Lin88] F. J. Lin, P. M. Chu, and M. T. Liu, "Protocol Verification Using Reachability Analysis: The State Space Explosion Problem and Relief Strategies," *proc. of the ACM SIGCOMM'87 Workshop*, pp. 126-135.

[Mue04] M. Zur Muehlen. Workflow-based Process Controlling: Foundation, Design and Application of Workflow-Driven Process Information Systems. Logos, Berlin, 2004.

[Ngu04] T. M. Nguyen, A. M. Tjoa, G. Kickinger, and P. Brezany, "Towards Service Collaboration Model in Grid-Based Zero Latency Data Stream Warehouse (GZLDSWH)," *proc. of the IEEE International Conference on Service Computing*, 2004.

[Nie92] M. Niezette and J. Stevenne, "An Efficient Symbolic Representation of Periodic Time," *Proc. of First International Conference on Information and Knowledge Management*, November 2-5, 1992.

[OMG03] OMG. Unified Modeling Language Specification: version 2.0. *Object Management Group Inc*. www.uml.org. 2003.

[Pil05] D. Pilone and N. Pitman. UML 2.0 in a Nutshell. O'Reilly Press, 2005.

[San94] R. Sandhu and P. Samarati, "Access Control: Principles and Practice," IEEE Communications Magazine, Vol. 32, No. 9, September 1994, pp. 40-48.

[San96] R. Sandhu, E. J. Coyne, H. L. Feinstein, C. E. Youman. Role Based Access Control Models, *IEEE Computer*, Vol. 29, No 2, February 1996.

[Sha05] B. Shafiq, J. Joshi, E. Bertino, and A. Ghafoor, "Secure Interoperation in a Multi-Domain Environment Employing RBAC Policies," *IEEE Transactions on Knowledge and Data Engineering,* Vol. 17, No. 11, pp. 1557-1577.

[She90] A. P. Sheth and J. A. Larson, "Federated Database Systems for managing Distributed, Heterogeneous, and Autonomous Databases," ACM Computing Surveys, Vol. 22, No. 3, September 1990, pp. 184-236.

[Tri03] A. Tripathi, T. Ahmed, and R. Kumar, "Specification of Secure Distributed Collaboration Systems," *proc of the IEEE International Symposium on Autonomous Distributed Systems (ISADS)*, pp. 149–156, 2003.

[Tri04] A. Tripathi, T. Ahmed, D. Kulkarni, R. Kumar and K. Kashiramka, "Context-Based Secure Resource Access in Pervasive Computing Environments," *proc*. of the *1st IEEE International Workshop on Pervasive Computing and Communications Security(IEEE PerSec'04)*, pp.159 – 163, 2004.

[Val79] L. G. Valiant, "The Complexity of Enumeration and Reliability Problems," *SIAM Journal on Computing*, Vo. 8, No. 3, August 1979, pp. 410-421.

[Wel03] V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman and S. Tuecke, "Security For Grid Services," *proc of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC'12)*, pp. 48-57, 2003.

[Wes89] C. H. West, "Protocol Validation in Complex Systems," *proc. Symposium on Communications Architecture and Protocols*, Austin, Texas, 1989, pp. 303-312,

[Wod96] D. Wodtke, J. Weissenfels, G. Weikum, and A. K. Dittrich, "The Mentor Project: Steps towards Enterprise-Wide Workflow Management," *proc. of the 12th IEEE International Conference on Data Engineering*, 1996.

[Yu04] J. Yu. Dynamic "Web Service Invocation based on UDDI," *proc of the IEEE International Conference on E-Commerce Technology for Dynamic E-Business*, pp. 154 – 157, 2004.

## Appendix

Proof of Proposition 1: The workflow composibility conditions WC2 and WC3 establish a partial ordering among all tasks of the PW. For any task pairs $\tau_i$ and $\tau_j \in$ PW such that $\tau_i$ precedes $\tau_j$ in task execution order $\phi_j^{\pi'} \geq \phi_i^{\pi'} + D$, where $0 < D < \infty$. Moreover, if the delay between completion of $\tau_i$ and $\tau_j$ is bounded by the interval $[T_{min}, T_{max}]$ then
$D_1 + \phi_i^{\pi'} \leq \phi_j^{\pi'} \leq D_2 + \phi_i^{\pi'}$ Where, $D_1 = T_{min} + D$, $D_2 = T_{min} + D$, and $D_2 > D_1 > 0$.

Note that the system of linear inequalities generated in step 1 of the task initiation time procedure does not have any constraint that bounds the sum of the initiation times of two or more tasks to a constant value. That is the system of linear inequalities generated in step 1 of the task initiation time procedure does not have any constraint of the form $M_1 \leq \sum_i \phi_i^{\pi'} \leq M_2$.

Therefore, minimizing the sum $\sum_i \phi_i^{\pi'}$ subject to the state residence time constraints and intra-domain workflow composibility constraints (WC1, WC2, and WC3) is equivalent to minimizing each individual $\phi_i^{\pi'}$. In other words, $\min(\phi_i^{\pi'})$ is the earliest time for initiation of task $\tau_i$ in state path $\pi$' such that the intra-domain workflow composibility conditions (WC1, WC2, and WC3) are satisfied. By the same argument, $\max(\phi_i^{\pi'})$ is the latest time for initiation of task $\tau_i$. □

Proof of Theorem 1: We will first prove this theorem for the case when $\tau_j$ has only one parent node and then we will prove the multi-parent case. In both cases we will use inductive reasoning.

Single Parent case ($\tau_j$ has only one parent task node in $G_X$):

*Base Case*: The task $\tau_1$ (first task of the PW) is the parent of $\tau_j$. The edge mapping function ensures that all the state transition paths in the path set $\prod_{1j}$ satisfies intra-domain workflow composibility condition WC1, WC2, and WC3 for task pair $\tau_1$ and $\tau_j$.

*Induction Step*: Suppose $\tau_i$ ($\tau_i \neq \tau_1$) is the parent task node of $\tau_j$ i.e., ($\tau_i$, $\tau_j$) $\in E[G_X]$. Let $\prod_{1i}$ denotes the set of all valid state paths from $\tau_1$ to $\tau_i$ that satisfy the intra-domain workflow composibility conditions WC1, WC2, and WC3 for task $\tau_i$ and all tasks that precede $\tau_i$ in the execution order of the PW. Let $\prod_{1j}$ be the set of all state paths from $\tau_1$ to $\tau_j$. The *for loop* in lines 2 -7 of the procedure *path-extend* called by *WPS* procedure ensures that for each $\pi_{1j} \in \prod_{1j}$, there exists a path $\pi_{1i} \in \prod_{1i}$ such that $\pi_{1i}$ is a prefix sub-path of $\pi_{1j}$, i.e., the path relation *start*($\pi_{1i}$, $\pi_{1j}$) holds.

The path set $\prod_{ij}$ returned by the edge-mapping procedure (in line 11 of *WPS* procedure), contains all paths that satisfy workflow composibility conditions WC1, WC2, and WC3 for the task pair $\tau_i$ and $\tau_j$. Consider a path $\pi_{ij} \in \prod_{ij}$. Suppose $\pi_i$ is a prefix path of $\pi_{ij}$ such that the path relations *finish*($\pi_i$, $\pi_{1i}$) and *start*($\pi_i$, $\pi_{ij}$) hold, where $\pi_{1i} \in \prod_{1i}$. The path $\pi_{ij}$ can be written as a concatenation of paths $\pi_i$ and $\pi$' i.e, $\pi_{ij} = \pi_i.\pi$'. In lines 5 and 6 of the *path-extend* procedure the state path $\pi_{1j}$ from $\tau_1$ to $\tau_j$ is computed by concatenating $\pi$' to the end of $\pi_{1i}$. $\boldsymbol{\pi_{1j} = \pi_{1i}.\pi'}$.

Since $\pi_{1j}$ include $\pi_{1i} \in \prod_{1i}$ as a sub-path, therefore all the intra-domain workflow composibility conditions that are true in $\pi_{1i}$ are also true in $\pi_{1j}$ for task $\tau_i$ and all tasks that precede $\tau_i$ in the execution order of the PW. $\pi_{1j}$ also includes $\pi_{ij} \in \prod_{ij}$ and therefore, $\pi_{1j}$ satisfies workflow composibility conditions WC1, WC2, and WC3 for the task pair $\tau_i$ and $\tau_j$. Moreover, in line 7 of the *path-extend* procedure, $\pi_{1j}$ is analyzed for satisfaction of workflow composibility WC1, WC2, and WC3 for all task pairs ($\tau_p$, $\tau_q$) such that ($\tau_p$, $\tau_q$) $\in$

$E[G_X]$ and $\tau_q = \tau_j$ or $\tau_q$ precedes $\tau_j$ in the task execution order. This proves that path $\pi_{1j}$ satisfies the two properties listed in the theorem statement.

Multiple parent case: We will prove the multi-parent case when $\tau_j$ is the first task node in $G_X$ that has $k$ parents with $k > 0$, as shown in Figure 11. The remaining multi-parent cases can be proved by similar reasoning.

*Base case*: $k = 1$, the proof is similar to the single parent case.

*Induction Step*: Let $\prod_{1j}^{(k-1)}$ be the set of all state paths that satisfy: (i) workflow composibility condition WC1 for task $\tau_j$ and all predecessor tasks of $\tau_j$ that can reach $\tau_j$ via the task nodes $\tau_i^{(1)},\ldots, \tau_i^{(k-1)}$. (ii) workflow composibility conditions WC2 and WC3 between task pairs $(\tau_i^{(1)}, \tau_j), \ldots.(\tau_i^{(k-1)}, \tau_j)$ and all other tasks pairs $(\tau_p, \tau_q)$ such that $(\tau_p, \tau_q) \in E[G_X]$ and $\tau_q$ precedes $\tau_i^{(1)},\ldots, \tau_i^{(k-1)}$ in the task execution order.

Let $\prod_{1j}^{(k)}$ be the set of all state paths that satisfy: (iii) workflow composibility condition WC1 for task $\tau_j$ and all predecessor tasks of $\tau_j$ that can reach $\tau_j$ via the task nodes $\tau_i^{(k)}$. (iv) workflow composibility conditions WC2 and WC3 between task pairs $(\tau_i^{(k)}, \tau_j)$ and all other tasks pairs $(\tau_p', \tau_q')$ such that $(\tau_p', \tau_q') \in E[G_X]$ and $\tau_q'$ precedes $\tau_i^{(k)}$ in the task execution order.

In the *WPS* procedure, the path set $\prod_{1j}$ is composed by taking an intersection of the path sets $\prod_{1j}^{(k-1)}$ and $\prod_{1j}^{(k)}$ (line 14). $\prod_{1j} = \prod_{1j}^{(k-1)} \cap \prod_{1j}^{(k)}$. Therefore, each path in the path set $\prod_{1j}$ satisfies (i), (ii), (iii), and (iv). Alternatively, each path $\pi_{1j} \in \prod_{1j}$ satisfies the following:

- Workflow composibility condition WC1 for task $\tau_j$ and all predecessor tasks of $\tau_j$ that can reach $\tau_j$ via its parent tasks.
- Workflow composibility conditions WC2 and WC3 between any task pair $(\tau_i, \tau_j)$ such that $(\tau_i, \tau_j) \in E[G_X]$.
- Workflow composibility conditions WC2 and WC3 between any task pair $(\tau_p, \tau_q)$ such that $(\tau_p, \tau_q) \in E[G_X]$ and $\tau_q$ precedes $\tau_j$ in the task execution order.

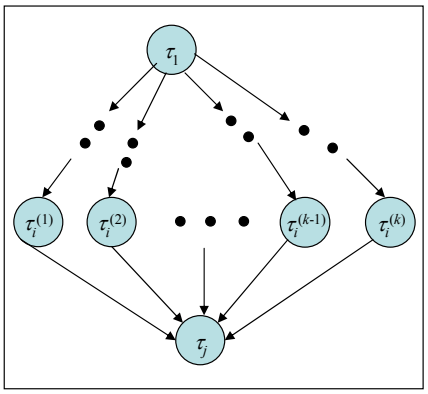


Figure 11

Proof of Theorem 2: The cross domain dependency verification procedure returns No, if there does not exist any path combination $(\pi_1, \pi_2,\ldots, \pi_n)$ (such that $\pi_k \in \Pi^{(k)}$ and $1 \le k \le n$), that satisfies the cross domain dependencies of the set $CS_{dep}$. Suppose on the contrary, that there exists a path combination $(\widehat{\pi_1}, \widehat{\pi_2}, \ldots, \widehat{\pi_n})$ that satisfies all the cross domain dependencies specified in the set $CS_{dep}$ and each $\widehat{\pi_k}$ satisfies the composibility conditions WC1, WC2, and WC3 for the projected workflow $PW_k$ assigned to $ID_k$. Since the verification procedure could not find the path combination $(\widehat{\pi_1}, \widehat{\pi_2}, \ldots, \widehat{\pi_n})$, the following two possibilities may occur:

1. At least one of the $\widehat{\pi_k}$ is not included in the path set $\Pi^{(k)}$, i.e, $\widehat{\pi_k} \notin \Pi^{(k)}$. As discussed in Section 4.2.1, the path set $\Pi^{(k)}$ returned by the procedure *WPS* is exhaustive and includes all the paths of $ID_k$ that satisfy the composibility conditions WC1, WC2, and WC3 for $PW_k$. Therefore, $\widehat{\pi_k} \notin \Pi^{(k)}$ implies that $\widehat{\pi_k}$ cannot support $PW_k$. Hence, the path combination $(\widehat{\pi_1}, \widehat{\pi_2}, \ldots, \widehat{\pi_n})$ cannot support the distributed workflow.

2. $\forall$ $k$, $\widehat{\pi}_k \in \Pi^{(k)}$. However, there exists at least one task $\tau_i \in \text{PW}_j$ such that $\phi_i^{\tilde{\pi}_j} \notin [\min(\phi_i^{\tilde{\pi}_j}), \max(\phi_i^{\tilde{\pi}_j})]$ or $\theta_i^{\tilde{\pi}_j} \notin [\min(\theta_i^{\tilde{\pi}_j}), \max(\theta_i^{\tilde{\pi}_j})]$, where $\phi_i^{\tilde{\pi}_j}$ and $\theta_i^{\tilde{\pi}_j}$ denote the initiation and completion times of $\tau_i$ in $\widehat{\pi}_j \in \Pi^{(j)}$. By Proposition 1, for any task $\tau_i \in \text{PW}_j$, $\min(\phi_i^{\pi})$, computed using the *task initiation time* procedure of Figure 5, is the earliest time instant at which $\tau_i$ can be initiated in a state path $\pi$ that satisfy composibility conditions WC1, WC2, and WC3 for $\text{PW}_j$. Similarly, $\max(\phi_i^{\pi})$ corresponds to the latest initiation time of task $\tau_i$. Since, $\widehat{\pi}_j \in \Pi^{(j)}$ and all paths in $\Pi^{(j)}$ satisfy WC1, WC2, and WC3 for $\text{PW}_j$, therefore, $\phi_i^{\tilde{\pi}_j} \in [\min(\phi_i^{\tilde{\pi}_j}), \max(\phi_i^{\tilde{\pi}_j})]$. Similarly, we can prove that $\theta_i^{\tilde{\pi}_j} \in [\min(\theta_i^{\tilde{\pi}_j}), \max(\theta_i^{\tilde{\pi}_j})]$.

With reference to the system of constraints (II) – (VIII) for cross-domain dependency verification, the above implies that all state paths in the combination $(\widehat{\pi}_1, \widehat{\pi}_2, \ldots, \widehat{\pi}_n)$ satisfy constraints (II) – (VII). Since the verification procedure returned No, therefore, constraint (I) is not satisfied for at least one pair of cross-domain paths, say $\widehat{\pi}_i$ and $\widehat{\pi}_j$. This implies that $\widehat{\pi}_i$ and $\widehat{\pi}_j$ do not satisfy the cross-domain dependencies among the *component service*s of $\text{PW}_i$ and $\text{PW}_j$. Therefore, the path combination $(\widehat{\pi}_1, \widehat{\pi}_2, \ldots, \widehat{\pi}_n)$ cannot support the distributed workflow. □