Itinerary-Based Access Control for Mobile Tasks in Scientific Workflows

Zijiang Yang [1†*] Shiyong Lu [2†] Ping Yang [3]

1 Western Michigan University, Kalamazoo, Michigan, USA

2 Wayne State University, Detroit, Michigan, USA

3 Binghamton University, Binghamton, New York, USA

Abstract

Current scientific workflow models require datasets to be transferred from their source to the hosts where they can be processed. This seriously impedes data-intensive applications. In order to address this limitation, we propose a notion of mobile tasks that can move from their home hosts towards datasets and perform computation on the dataset side. Since a mobile task might migrate across various administrative domains and get executed at multiple hosts, it is critically important to ensure the security of a mobile-task-based workflow system. In this paper, we propose an itinerary-based access control model to ensure the secure migration of mobile tasks.

1 Introduction

Recently, scientific workflows have gained tremendous momentum due to their critical role in e-Science [8]. Scientists use scientific workflows to "glue" together data management, analysis, simulation and visualization services over often voluminous complex and distributed scientific data and services. In contrast to traditional business workflows, which are task-centric and control-flow oriented, scientific workflows are typically data-centric and dataflow-oriented, and thus pose different challenges [8].

In contrast to traditional business workflows [4, 6], a scientific workflow has the following two features: (1) Different tasks might be serviced by different providers and thus geographically distributed across different administrative domains. For example, while *Segmentation* might be serviced by John Hopkins University, *Registration* might be serviced by University of Michigan. Most traditional business workflows only consider tasks that are within one enterprise. (2) Voluminous complex and distributed scientific data need to be integrated with various tools to conduct a complicated scientific analysis. Each dataset is potentially large in size.

Together, these two features impose a new computational challenge over traditional workflow engines: since tasks are static in traditional workflows, large datasets need to be transferred from source hosts to target hosts where computational tasks reside, resulting in extremely unbearable network communication overhead.

To overcome this limitation, we propose a mobile task model for scientific workflows where mobile tasks can migrate from one host to another towards large datasets to conduct data-intensive computation. More specifically, each mobile task is equipped with an *itinerary*, the set of hosts that the mobile task will visit and the pattern of visiting them. However, since a mobile task might migrate across several administrative domains and get executed at multiple hosts, it is critically important to develop trustworthy mechanisms to ensure the secure migration and execution of these itinerary-driven mobile tasks.

The main contributions of this paper are: (1) We propose a mobile task model for scientific workflows to meet the need of data-intensive applications. (2) We design a formal itinerary-based access control model. While our previous work [15] considers only the itinerary of a mobile task, both the visit history and future itinerary of a mobile task are considered in this paper. (3) We propose formal syntax and semantics of itinerary based access control policy, and develop algorithms to verify the access request of a mobile task against the access control policy at the host. (4) We develop a host visit scheduling algorithm for mobile tasks based on their itineraries and the dynamic host visit scenarios.

Organization. The rest of the paper is organized as follows. Section 2 presents an overview of related work. Section 3 describes our proposed architecture for mobile task migration and execution. The algorithms to schedule a mobile task based on its itinerary is presented in Section 4, followed by the innovative approaches on host access control in Section 5. Finally, Section 6 concludes the paper and suggests some possible future work.

[*] This work was supported by funds from the Faculty Research and Creative Activities Support Fund, Western Michigan University.

[†] The first two authors contributed equally to this paper.

2 Related work

In recent years, scientific workflows have gained great momentum due to their critical roles in e-Science and cyberinfrastructure applications [8]. There is a plethora of scientific workflows covering a wide range of scientific disciplines. Yu and Buyya characterize and classify various approaches for building and executing workflows on the Grid [18] and highlight that scheduling workflow tasks in a distributed Grid environment is a challenging problem. The workflow scheduling problem is showcased in the ASKAON project [14] by Wieczorek et al. While Simmhan, Plale and Gannon argue that data provenance is a critical component of scientific workflows and summarize the key research efforts and open problems in this area [11], Mederios et al. envision that shared catalogues of workflows indexed by metadata should be used in order to facilitate the reuse of scientific workflows in new applications [9]. Nevertheless, none of the above work has addressed the security and correctness issues of scientific workflows that support mobile tasks.

Many researchers have investigated various security issues in mobile agents [12, 2], in particular, the access control at each host [1, 10]. However, existing host visit access control models only consider the visit history of a mobile agent and do not consider future behavior of a mobile agent. In contrast, our access control model supports the specification of a host visit access control policy that not only considers the host visit history of a mobile agent but also its future itinerary which prescribes its future mobility behavior. Another unique feature of our model is that each mobile task is equipped with a "scheduler", which is able to communicate with the list of candidate hosts that the mobile task intends to visit according to its itinerary and then schedules the next host to visit. The next host will guarantee the permission of such an access. As a result, a mobile task will always follow a trace of hosts which permit its access if such an trace exists. None of existing work supports this salient scheduling feature.

Finally, our notion of *mobile tasks* is developed from our previously proposed notion of *itinerary-driven mobile agents* [7, 5, 15, 16] with the following additional features necessary for scientific workflow applications: (i) mobile tasks have well-defined input and output ports such that data links can be used to connect these ports to compose composite task or a scientific workflow, while traditional mobile agents do not support input and output ports that target for data links, and (ii) a mobile task might possess some of the ACID (Atomicity, Consistency, Isolation, and Durability) properties of a transaction, while traditional mobile agents usually do not support such properties.

3 Architecture for Mobile Task Migration and Execution

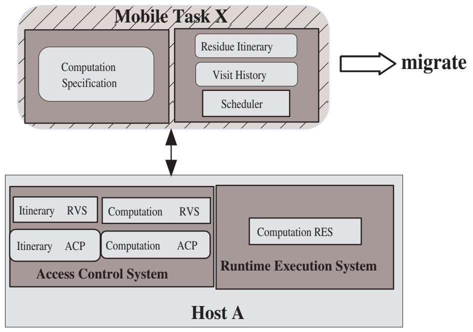

Figure 1. Architecture for mobile task migration and execution

Figure 1 depicts our proposed architecture for the migration and execution of mobile tasks at a host. A mobile task X consists of computation and itinerary components, where specification of the navigational behavior of a mobile task is separated from the specification of its computational behavior.
-*Computation Component.* The computation specification can be written in a traditional programming language such as Java.
-*Itinerary Component.* An itinerary includes visit history and residue itinerary. The visit history records all the hosts that this mobile task has visited, and the residue itinerary is a host pattern to be visited in the future. The scheduler considers both the visit history and residue itinerary and interacts with access control systems of hosts to schedule which host H to visit next. Once the access request to H is granted, X will migrate to H and the residue itinerary and visit history will be updated accordingly.

Each host is equipped with a mobile task virtual machine to support the secure execution of mobile tasks. The virtual machine consists of the following two components.
-*Runtime Execution System (RES).* The RES will execute the computation task specified in the computation component of a mobile task.
-*Access Control System (ACS).* The ACS of a host contains itinerary Access Control Policy (ACP) and computation ACP that are used to control the host visit and resource access privileges of mobile tasks. When a host A receives a visit request from a mobile task X, X's itinerary will be verified against A's itinerary ACP. X may visit A only if X's itinerary satisfies A's itinerary ACP. After X migrates to A, A's computation Runtime Verification System (RVS) will verify at runtime if X's computation model satisfies host A's computation ACP. If it does, then agent X can execute

its computation specification; otherwise, none of agent X's computation specification will be executed.

In this paper we concentrate on mobile task scheduling and itinerary access control. Since the computation task of a mobile task can be developed using general purpose programming language, software verification techniques [3, 13, 17] can be applied to handle computation access control.

4 Mobile Task Scheduling

We consider a simple form of itinerary that has the following BNF syntax.

$$i ::= s \mid i_1 ; i_2 \mid i_1 \# i_2 \mid i_1 \parallel i_2$$

where ;, $\#$, and $\parallel$ denote sequential, nondeterministic choice, and parallel operators, respectively. We use the interleaving semantics for the parallel operator.

Given an itinerary I, Algorithm 1 calculates the set of hosts to be visited next and the corresponding residue itineraries. It is a migration set because there maybe multiple hosts that can be visited following the specification of an itinerary. Each item in the returned set is a pair $P = (s_k, i_k)$ where s_k, referred as $P.h$, is a host that a mobile task may visit next, and i_k, referred as $P.r$, is the residue itinerary if the mobile task chooses to visit s_k next. The algorithm works as follows. If the itinerary I is s (Lines 1-2), the host to be visited next is s. If I has the top level pattern $i_1; i_2$, the algorithm first obtains the set M from i_1. For each item in M that is not $(\_, \emptyset)$ where $\_$ is a wildcard, i_2 is appended to the item's residue itinerary (Lines 6-7); otherwise, the residue itinerary becomes i_2 in the item(Line 9). If I is $i_1 \# i_2$ (Lines 12-13), the migration set is the disjunction of those of i_1 and i_2. Finally, if I is $i_1 \parallel i_2$, the migration sets for both i_1 and i_2 need to be considered. The residue itineraries of i_1 are interleaved with i_2, and the residue itineraries of i_2 are interleaved with i_1.

The procedure *Schedule()*, shown in Algorithm 2, is used to schedule which host to visit next for a mobile task. The parameter `history`, initially $\emptyset$, is the list of hosts that the mobile task has visited. The parameter `residue`, initially the full itinerary I, contains the hosts to be visited. The parameter `stack`, initially empty, is a stack with each item recording other possible route at each traversal step. At line 1 the function *Migration()* is called to compute the migration set M. The loop between lines 2-23 guides the task based on M and `stack`. The loop terminates when either the mobile task has successfully traversed its full itinerary, or has exhausted all alternative routes but still cannot proceed as required. The while loop between lines 3-16 tries to navigate the mobile task based on the migration set M. A pair $(next, R)$ is removed from M where $next$ is the host to be visited next, and R is the residue after $next$ is visited.

Algorithm 1 SET MIGRATION(ITINERARY I)

```
 1: if $I = s$ then
 2:    $M = \{(s, \emptyset)\}$;
 3: else if $I = i_1; i_2$ then
 4:    $M$ = Migration($i_1$);
 5:    for all $(P \in M)$ do
 6:       if $P \neq (\_, \emptyset)$ then
 7:          $P = (P.h, P.r; i_2)$;
 8:       else
 9:          $P = (P.h, i_2)$;
10:       end if
11:    end for
12: else if $I = i_1 \# i_2$ then
13:    $M$ =Migration($i_1$)$\cup$ Migration($i_2$);
14: else if $I = i_1 \parallel i_2$ then
15:    $M_1$ = Migration($i_1$);
16:    $M_2$ = Migration($i_2$);
17:    $M = \{M_1.h, M_1.r \parallel i_2\} \cup \{M_2.h, M_2.r \parallel i_1\}$;
18: end if
19: return $M$;
```

An access request to host $next$ is made at Line 9. The algorithm for host access control is discussed in Section 5. If the access request is denied, another pair will be selected from M. Otherwise the mobile task will visit $next$ and continue its scheduling based on its new history and residue itinerary (Lines 10-15). Note that at Line 11 we save M in `stack` for the purpose of backtrack in the future. This is because the access of host $next$ does not guarantee that the mobile task can visit the hosts specified in R successfully. It is possible that the mobile task has to backtrack to choose a different route. M becomes empty when accesses to all potential next hosts are denied. In this case, the mobile task will backtrack. If `stack` is empty (Lines 17-19), the mobile task has exhausted all possible routes that conform to the mobile task's residue itinerary, but still cannot fulfill the requirement of the itinerary. Otherwise, the mobile task will pop a set to replace M and backtrack to the last host in its `history` (Lines 20-22).

5 Itinerary-Based Access Control

In this section we discuss approaches for itinerary-based host access control. We first define the syntax of the access control policy in Section 5.1. Then we explain its semantics using Host Transition Graph in Section 5.2. Finally in Section 5.3, we present algorithms to verify mobile task's itinerary against the access control policy.

5.1 Access Control Policy

We consider the Computational Tree Logic (CTL) as the basis for the itinerary-based Access Control Policy (ACP). In CTL, formulas are composed of *path quantifiers* that are used to describe the branching structure, and *temporal operators* that are used to describe properties of a path. There are

Algorithm 2 SCHEDULE(STACK, HISTORY, RESIDUE)

```
1: $M$ =Migration(residue);
2: while true do
3:    while $M \neq \emptyset$ do
4:       if $(s, \emptyset) \in M$ then
5:          visit s;
6:          terminate with success;
7:       end if
8:       $(next, R)$ =Remove($M$);
9:       $r$ =AskAccessPermission(next, history, $R$);
10:      if $r == OK$ then
11:         stack.Push($M$);
12:         Visit $next$ and perform computation task;
13:         history.appendEnd($next$);
14:         Schedule(stack, history, $R$);
15:      end if
16:   end while
17:   if stack is empty then
18:      terminate with failure;
19:   end if
20:   $M$ = stack.Pop( );
21:   $h$ = history.removeEnd( );
22:   backtrack to $h$;
23: end while
```

two path quantifiers: A (for all paths) and E (for some path); and five temporal operators: X(next time), F(eventually), G(always), U(until). In order to reason about past-time behaviors, we introduce the following past temporal operators: Y(in the previous time instance), and S (since). Note that in our context, the visit history has only one path without any branches, so the past temporal operators can only be combined with A.

Formally, the syntax of an ACP formula is defined by the grammar below. In the definition, ϕ defines an ACP formula that builds upon its past-time formula ν and future-time formula μ.

Definition 1. [Access Control Policy]

$$\phi \; ::= \; \mu \mid \nu \mid \phi \vee \phi \mid \neg\phi \qquad (1)$$

$$\mu \; ::= \; h \mid \mathsf{EX}\,\mu \mid \mu\,\mathsf{EU}\,\mu \qquad (2)$$

$$\nu \; ::= \; h \mid \mathsf{AY}\,\nu \mid \nu\,\mathsf{AS}\,\nu \qquad (3)$$

where h is a host.

Note that the unary temporal connective EX (*possibly-next*), AY (*previous*), and the binary temporal connective EU (*possibly-until*, AS (*since*)can be used to define other connectives:
- *possibly-eventually*: $EF\mu$ for $true$ EU μ;
- *Inevitably-next*: AX μ for ¬ EX ¬μ;
- *Inevitably-always*: AF μ for ¬ EF ¬μ.
- *past*: AP ν for $true$ AS ν;
- *Inevitably-past*: AH ν for ¬ AP ¬ν.

For example, AP$(f \wedge$ AP $e)$ specifies that a mobile task can visit current host only if it has visited e first and then visited f sometime later (possibly visit other hosts in between).

5.2 Host Transition Graph

We define the semantics of access control policy with respect to a Host Transition Graph (HTG), defined as a tuple $G = \langle V, T, \lambda, c\rangle$ where V is a set of vertices, $T \in H \times H$ is a set of transitions, λ is a function that labels each vertex with a set of host control policy formulas, and c is the to-be-visit-next vertex. In order to link different components during HTG construction, we define two sets $START$ and END. Given a subgraph $G' \subseteq G$, $s \in START(G')$ iff $s \in G'$ and $\exists(t \notin G' \wedge (t \rightarrow s) \in G)$, and $s \in END(G')$ iff $s \in G'$ and $\exists(t \notin G' \wedge (s \rightarrow t) \in G)$.

Algorithm 3 CREATHTG(HISTORY, h_0, RESIDUE)

```
1: $G_H$ = CreateHistoryHTG(history);
2: $G_R$ = CreateResidueHTG(residue);
3: create vertex $v_0$ with $\lambda(v_0) = \{h_0\}$;
4: $G = G_H \cup G_R \cup (END(G_H) \rightarrow v_0) \cup (v_0 \rightarrow START(G_R))$;
5: $G.c = v_0$;
```

Algorithm 3 shows the pseudo-code on how to construct a HTG based on three inputs: `history` is the hosts that has been visited, h_0 is the host to be visited next, and `residue` is the residue itinerary. The algorithm first creates the HTGs G_H and G_R for `history` and `residue` (Lines 1-2), then link the two components through the to-be-visit-next vertex v_0 that is labeled with the set $\{h_0\}$.

Algorithm 4 shows how to construct the HTG component for `residue`. It is straightforward when `residue` is a single host (Lines 1-3). In case `residue` is sequential (Lines 4-9), we first construct HTGs G_1 and G_2 for its component i_1 and i_2, then the edges link the end set of G_1 and the start set of G_2 is added. Note that $(END(G_1) \rightarrow START(G_2))$ should be interpreted as Cartesian product with direction. For example, if $END(G_1) = \{s_1, s_2\}$ and $START(G_2) = \{s_3, s_4\}$, following four edges are added: $s1 \rightarrow s3, s2 \rightarrow s3, s1 \rightarrow s4, s2 \rightarrow s4$. In the case of non-deterministic choice, the start and end sets are the union of its components. Finally when `residue` is an interleaving itinerary, we need to consider all the combinations. Each pair of edge $s_1 \rightarrow s_2$ and $t_1 \rightarrow t_2$ in different components result in Cartesian product $s_1 \rightarrow s_2 \times t_1 \rightarrow t_2$, which denote the following edges: $s_1 \rightarrow s_2 \rightarrow t_1 \rightarrow t_2, t_1 \rightarrow t_2 \rightarrow s_1 \rightarrow s_2, s_1 \rightarrow t_1 \rightarrow s_2 \rightarrow t_2, t_1 \rightarrow s_1 \rightarrow t_2 \rightarrow s_2, t_1 \rightarrow s_1 \rightarrow s_2 \rightarrow t_2, s_1 \rightarrow t_1 \rightarrow t_2 \rightarrow s_2$. The start(end) set of the graph needs to be re-calculated, since the start(end) point of a component may no longer be the start(end) point of the resulted HTG. The algorithm to calculate the new start(end) set is omitted here.

The semantics of access control policy can be explained on host transition graphs. When a model task X requests for access at host h_0, the itinerary tuple $\langle hisotry, h_0, residue\rangle$ of X will be used to construct a HTG G_X. Note that initially each vertex in G_X is labeled with a singleton set and

Algorithm 4 HTG CREATRESIDUEHTG(RESIDUE)

```
1:  if Residue == $s$ then
2:      add a vertex $v$ to $G$ with $\lambda(v) = \{h\}$;
3:      $START(G) = END(G) = \{v\}$;
4:  else if Residue == $i_1; i_2$ then
5:      $G_1$ = CreatResidueHTG($i_1$);
6:      $G_2$ = CreatResidueHTG($i_2$);
7:      $G = G_1 \cup G_2 \cup (END(G_1) \rightarrow START(G_2))$;
8:      $START(G) = START(G_1)$;
9:      $END(G) = END(G_2)$;
10: else if Residue == $i_1 \# i_2$ then
11:     $G_1$ = CreatResidueHTG($i_1$);
12:     $G_2$ = CreatResidueHTG($i_2$);
13:     $G = G_1 \cup G_2$;
14:     $START(G) = START(G_1) \cup START(G_2)$;
15:     $END(G) = END(G_1) \cup END(G_2)$;
16: else if Residue == $i_1 || i_2$ then
17:     $G_1$ = CreatResidueHTG($i_1$);
18:     $G_2$ = CreatResidueHTG($i_2$);
19:     for all $s_1 \rightarrow s_2 \in G_1$ do
20:         for all $t_1 \rightarrow t_2 \in G_2$ do
21:             $G = G \cup \{s_1 \rightarrow s_2 \times t_1 \rightarrow t_2\}$;
22:         end for
23:     end for
24:     $START(G)$ = GetStartSet($G$);
25:     $END(G)$ = GetEndSet($G$);
26: end if
27: return G;
```

$G_X.c$ is labeled with $\{h_0\}$. It is also possible that multiple vertices in G_X are labeled with the same host name. The set of access control policy formulas labeled at each vertex will be changed in the algorithms introduced in Section5.3.

The definition on whether G satisfies the access control policy formula ϕ is defined as follows.

Definition 2. [Satisfaction relationship $\models$] *Let v_0 be a vertex in G_X created from the itinerary of mobile X, and ϕ be an access control policy. The relation $v_0 \models \phi$ is defined inductively as follows:*
-$v_0 \models s$ iff $s \in \lambda(v_0)$.
-$v_0 \models \neg\phi$ iff not $v_0 \models \phi$.
-$v_0 \models \phi_1 \vee \phi_2$ iff $v_0 \models \phi_1$ or $v_0 \models \phi_2$.
-$v_0 \models \mathsf{EX}\,\phi$ iff for some vertices v such that $(v_0 \rightarrow h) \in G, v \models \phi$.
-$v_0 \models \phi_1\,\mathsf{EU}\,\phi_2$ iff for some paths $(v_0, v_1, \ldots)$, $\exists i[i \geq 0 \wedge v_i \models \phi_2 \wedge \forall j[0 \leq j < i \implies v_i \models \phi_1]]$
-$v_0 \models \mathsf{AY}\,\phi$ iff for the vertex v such that $(v, v_0) \in R, v \models \phi$.
-$v_0 \models \phi_1\,\mathsf{AS}\,\phi_2$ iff for the path $(\ldots, v_1, v_0)$, $\exists i[i \geq 0 \wedge v_i \models \phi_2 \wedge \forall j[0 \leq j < i \implies v_i \models \phi_1]]$
The mobile task X can access host h with control policy ϕ iff $G.c \models \phi$.

5.3 Access Control Verification

In this section we discuss the algorithms to verify a host transition graph G submitted by a mobile task against an access control policy ϕ at a host h_0. We consider only EX, EU, AP, AS as other operators can be defined from these basic connectives.

In the proposed algorithm we proceed inductively on the structure of ϕ. The subformulas of ϕ is defined as follows.

Definition 3. [Subformulas] *The set $Sub(\phi)$ of subformulas of ϕ is defined inductively:*

$$\begin{array}{rcl} Sub(h) & = & \{h\} if\ h\ is\ a\ host \\ Sub(\phi_1 \vee \phi_2) & = & \{\phi_1 \vee \phi_2\} \cup Sub(\phi_1) \cup Sub(\phi_2) \\ Sub(\neg\phi) & = & \{\neg\phi\} \cup Sub(\phi) \\ Sub(\mathsf{EX}\,\phi) & = & \{\mathsf{EX}\,\phi\} \cup Sub(\phi) \\ Sub(\phi_1\,\mathsf{EU}\,\phi_2) & = & \{\phi_1\,\mathsf{EU}\,\phi_2\} \cup Sub(\phi_1) \cup Sub(\phi_2) \\ Sub(\mathsf{AY}\,\phi) & = & \{\mathsf{AY}\,\phi\} \cup Sub(\phi) \\ Sub(\phi_1\,\mathsf{AS}\,\phi_2) & = & \{\phi_1\,\mathsf{AS}\,\phi_2\} \cup Sub(\phi_1) \cup Sub(\phi_2) \end{array}$$

$OrderedSub(\phi)$ is a queue with the subformulas of ϕ such that a formula appears only after all its subformulas. That is, if $\phi_1 \in Sub(\phi)$ and $\phi_2 \in Sub(\phi_1)$, then ϕ_2 precedes ϕ_1 in $OrderedSub(\phi)$.

Definition 4. [Characteristic Region] *Given a (sub)formula ϕ, the characteristic region $[\phi]_G$ of ϕ in G is the set of all the vertices that satisfy ϕ. Let $\lambda(v)$ be the set of formulas that are labeled in v, then $v \in [\phi]_G \leftrightarrow \phi \in \lambda(v)$.*

Algorithm 5 BOOLEAN ACCESSCONTROL(HTG G, POLICY ϕ)

```
1:  for all $\psi \in OrderedSub(\phi)$ do
2:      if $\psi \equiv (\phi_1 \vee \phi_2)$ then
3:          AccessControlOr($G$, $\phi_1 \vee \phi_2$);
4:      else if $\psi \equiv (\neg\phi_1)$ then
5:          AccessControlNot($G$, $\neg\phi_1$);
6:      else if $\psi \equiv (\mathsf{EX}\,\phi_1)$ then
7:          AccessControlEX($G$, $\mathsf{EX}\,\phi_1$);
8:      else if $\psi \equiv (\phi_1\,\mathsf{EU}\,\phi_2)$ then
9:          AccessControlEU($G$, $\phi_1\,\mathsf{EU}\,\phi_2$);
10:     else if $\psi \equiv (\mathsf{AY}\,\phi_1)$ then
11:         AccessControlAY($G$, $\mathsf{AY}\,\phi_1$);
12:     else if $\psi \equiv (\phi_1\,\mathsf{EU}\,\phi_2)$ then
13:         AccessControlAS($G$, $\phi_1\,\mathsf{AS}\,\phi_2$);
14:     end if
15: end for
16: return($\phi \in \lambda(G.c)$? Yes : No);
```

In order to check if the HTG G satisfies the access control policy ϕ at host h_0, we compute the characteristic region on ϕ's ordered subformulas inductively, as shown in Algorithm 5. The for loop (Lines 4-18) iterates over all the subformulas of ϕ and call functions that handle particular formula types. After the loop, each vertex v is labeled by a set $\lambda(v)$ of subformulas of ϕ that satisfies v. Note that the policy ϕ is its own subformula and the last item in $OrderedSub(\phi)$. Finally (Line 19), if the input policy ϕ

is a member of $\lambda(G.c)$, the host h_0 should grant access to the mobile task who submits the request; otherwise h_0 will reject the request. This is because $G.c \models \phi$ iff $h_0 \in [\phi]_G$.

Algorithm 6 ACCESSCONTROLEX(HTG G, POLICY $\mathsf{EX}\,\phi_1$)

```
1: for all $v \in G$ do
2:     if $\exists v' | (v \rightarrow v') \in G \wedge \phi_1 \in \lambda(v')$ then
3:         $\lambda(v) = \lambda(v) \cup \mathsf{EX}\,\phi_1$;
4:     end if
5: end for
```

Algorithm 6 shows the function to check formulas with format $\mathsf{EX}\,\phi_1$. For each $v \in G$, if one of its successor v' has $\phi_1 \in \lambda(v')$, then $\mathsf{EX}\,\phi_1 \in \lambda(v)$ due to the semantics of EX. The algorithms to check other operators can be designed similarly.

Theorem 1. *Let ϕ be an access control policy with ρ symbols, and let G be a host transition graph with n hosts and m transitions. Given the input ϕ and G, Algorithm 5 solves the access control problem in $O(\rho \times (n + m))$ time, and requires $O(\rho \times n) space$.*

6 Conclusions and Future Work

We have presented an architecture for mobile tasks to address the need for supporting data-intensive applications in the context of scientific workflows. To support secure migration and execution of mobile tasks, we proposed an itinerary based access control model for host visit that not only considers the host visit history of a mobile task but also its future migration behavior that is prescribed by the residue itinerary.

Several future work can be pursued. First, itinerary language can be extended with additional constructs such as cloning and loop. Second, the access control model can be extended to support fine-grained access to scientific datasets. Finally, the mobile task framework can be extended for collaborative scientific workflows, in which a consortium can be formed by several member institutions for a collaborative scientific study.

References

[1] C. Cao and J. Lu. A path-history-sensitive access control model for mobile agent environment. In *ICDCSW '05: Proceedings of the Third International Workshop on Mobile Distributed Computing (MDC) (ICDCSW'05)*, pages 660–663, Washington, DC, USA, 2005. IEEE Computer Society.

[2] W. M. Farmer, J. D. Guttman, and V. Swarup. Security for mobile agents: Authentication and state appraisal. In *Proceedings of the Fourth European Symposium on Research in Computer Security*, pages 118–130, Rome, Italy, 1996.

[3] F. Ivančić, Z. Yang, I. Shlyakhter, M. Ganai, A. Gupta, and P. Ashar. F-SOFT: Software verification platform. In *Computer-Aided Verification*, pages 301–306. Springer-Verlag, 2005. LNCS 3576.

[4] S. Lu. *Semantic Correctness of Transactions and Workflows*. PhD thesis, State University of New York at Stony Brook, May 2002. Advisor: Dr. Arthur Bernstein.

[5] S. Lu. Itinerary safety reasoning and assurance. In C. zhong Xu, editor, *Scalable and Secure Internet Services and Architecture*, pages 247–262. Chapman & Hall/CRC, 2005.

[6] S. Lu, A. Bernstein, and P. Lewis. Completeness and realizability: Conditions for automatic generation of workflows. *International Journal of Foundations of Computer Science*, 17(1):223–245, 2006.

[7] S. Lu and C. zhong Xu. A formal framework for mobile agent itinerary specification, safety reasoning, and logic analysis. In *Proc. of the 3rd IEEE International Workshop on Mobile Distributed Computing (MDC05), in conjunction with ICDCS2005*, Columbus, OH, USA, 2005.

[8] B. Ludascher and C. Goble. Guest editor's introduction to the special section on scientific workflows. *SIGMOD Record*, 34(3):3–4, Sept. 2005.

[9] C. B. Medeiros, J. Perez-Alcazar, L. Digiampietri, G. Z. Pastorello, A. Santanche, R. S. Torres, E. Madeira, and E. Bacarin. WOODSS and the web: Annotating and reusing scientific workflows. *SIGMOD Record*, 34(3):18–23, Sept. 2005.

[10] G. Navarro, J. Borrell, J. A. Ortega-Ruiz, and S. Robles. Access control with safe role assignment for mobile agents. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 1235–1236, New York, NY, USA, 2005. ACM Press.

[11] Y. Simmhan, B. Plale, and D. Gannon. A survey of data provenance in e-science. *SIGMOD Record*, 34(3):31–36, Sept. 2005.

[12] J. Tardo and L. Valente. Mobile agent security and Telescript. In *IEEE CompCon '96*, pages 58–63, 1996.

[13] C. Wang, Z. Yang, F. Ivancic, and A. Gupta. Disjunctive image computation for emebedded software verification. In *Design, Automation and Test in Europe (DATE'06)*, Munich, Germany, Mar. 2006. to appear.

[14] M. Wieczorek, R. Prodan, and T. Fahringer. Scheduling of scientific workflows in the ASKALON grid environment. *SIGMOD Record*, 34(3):57–62, Sept. 2005.

[15] Z. Yang, S. Lu, and P. Yang. Runtime security verification for itinerary-driven mobile agents. In *Proc. of the 2nd IEEE International Symposium on Dependable, Autonomic and Secure Computing*, pages 177–186, Indianapolis, USA, September 2006.

[16] Z. Yang, S. Lu, and P. Yang. Model checking approach to itinerary-based access control enforcement of mobile tasks in scientific workflows. *Journal of Autonomic and Trusted Computing*, 2007. To appear.

[17] Z. Yang, C. Wang, F. Ivancic, and A. Gupta. Mixed symbolic representations for model checking software programs. In *ACM/IEEE International Conference on Formal Methods and Models for Codesign (Memocode'06)*, 2006.

[18] J. Yu and R. Buyya. A taxonomy of scientific workflow systems for grid computing. *SIGMOD Record*, 34(3):44–49, Sept. 2005.