Trustworthy and Dynamic Mobile Task Scheduling in Data-Intensive Scientific Workflow Environments

Zijiang Yang[a], Shiyong Lu[b], Ping Yang[c], Andrey Kashlev[b]

[a]Western Michigan University, Kalamazoo, Michigan, USA

[b]Wayne State University, Detroit, Michigan, USA

[c]Binghamton University, Binghamton, New York, USA

Abstract

There is an increasing demand for data-intensive applications in which scientists use scientific workflows to integrate together data management, analysis, simulation and visualization services over often voluminous complex and distributed scientific data and services. One major limitation of current scientific workflow models is that each workflow task is stationary, requiring a dataset to be transferred from its source host to a target host where a stationary task resides before a computation can be performed on the dataset. This limitation seriously impedes data-intensive applications since it can take an unbearable amount of time to transfer large amount of datasets from their sources to the host where a stationary task resides. In order to address this limitation, in this paper, we apply the idea of mobile agents to distributed scientific workflows. In contrast to stationary tasks, mobile tasks move from their home hosts towards datasets and perform computation on the dataset side. Since in data-intensive applications, it is often the case that the size of a mobile task is much smaller than the size of a dataset, our mobile-task approach can greatly reduce the network communication overhead. 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. The lack of effective access control model creates security hole in distributed scientific workflows. In this paper, we address this problem using an itinerary-based access control model and a host visit scheduling algorithm that prevent arbitrary tasks from accessing and being executed on the current host.

Keywords: scientific workflow, mobile task, access control, formal verification

1. Introduction

Scientific workflows play a critical role in e-Science [1]. They are essential to integrate data management, analysis, simulation and visualization services over 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 new challenges [1].

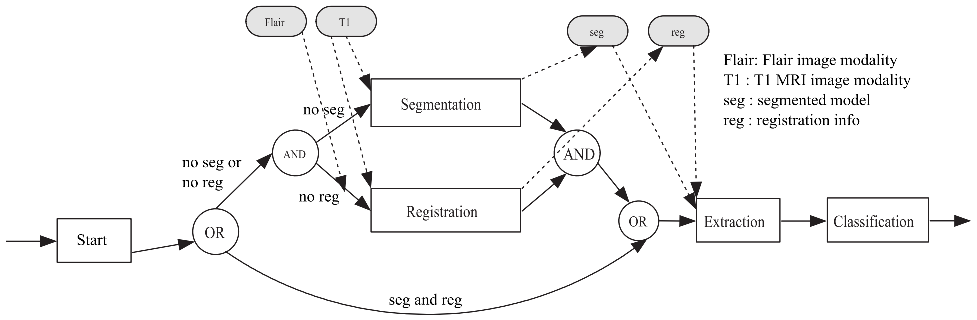


Figure 1: A sample scientific workflow in pediatric epilepsy.

A sample scientific workflow in pediatric epilepsy is shown in Figure 1, which is taken from our ongoing pediatric epilepsy project. This workflow integrates various medical imaging processing tools (such as *Segmentation* and *Registration*), analysis tools (such as *Extraction* and *Classification*), and various distributed multimodality neuroimaging data (such as Fair and T1) to perform integrative analysis to facilitate the identification of the locations of second epileptic foci in children's brains. The workflow contains both control flows and data flows, which are represented by solid edges and dashed edges, respectively. Tasks *Segmentation* and *Registration* will be executed concurrently, but their execution is activated only when there exists no segmented model or registration information, respectively. This is specified by the conditions labeled over the incoming edges of *Segmentation* and *Registration* – *no seg* and *no reg*; however, if both segmented model and registration information are available, then the execution of task *Extraction* can be activated immediately after the execution of *Start*. In terms of data flows for this specific workflow, task *Segmentation* takes T1 MRI image modality (T1) as input and produces the segmented model as output; task *Registration* takes both Flair image modality (Flair) and T1 MRI image modality (T1) as inputs and produces the registration information (reg) as output; then, both the segmented model and registration information are the inputs of task *Extraction*.

In contrast to traditional business workflows [2, 3], this example illustrates the following features of a scientific workflow:

– 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 organization.
– 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.

For example, for each patient, each modality image dataset typically consists of 30 to 40 images, with each image of the size of several Megabytes.

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. Our task and access control models enable secure execution of scientific workflow inside grid and cloud environments.

To motivate the need for an itinerary-based access control policy, consider the following access control policy examples, in which a mobile task implements several light-weight services but will perform these services over a distributed set of datasets.

Example 1.1

1. In a brain DTI analysis application, a mobile task needs to visit *e* first to perform a fiber tracking service, and then perform a fiber analysis service at current host *h* (Figure 2a).

2. In a temporal lobe epilepsy surgical candidacy determination application, segmentation and registration services need to be performed first at hosts *e* and *f*, respectively, in any order before the extraction service can be performed at current host *h* (Figure 2b).
3. In an epilepsy second foci location application, the cortical element parcelation service has to be performed first at host *e*, and then a fiber tracking service has to be performed at host *f* before the second foci location analysis service can be performed at current host *h* (Figure 2c).

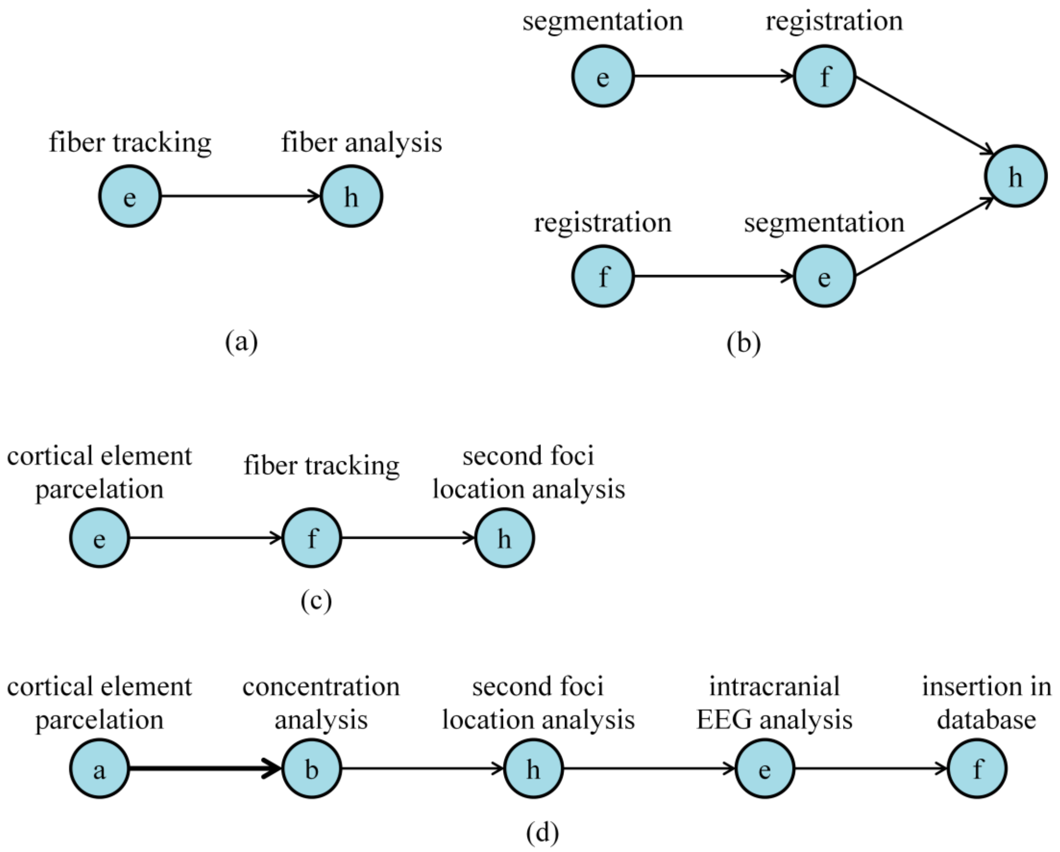


Figure 2: An example of access control policy: (a) DTI analysis application; (b) Temporal lobe epilepsy surgical candidacy determination application; (c) Epilepsy second foci location application; (d) Epilepsy intracranial EEG surgery application – bold arrow indicating no intermediate hosts can be visited between *a* and *b*.

4. In an epilepsy intracranial EEG surgery application, a mobile task has to visit hosts *a* and *b* to perform cortical element parcelation and concentration analysis in sequence first before the mobile task can perform a concentration comparison at the current host *h*. If the patient is identified as a surgery candidate, then the mobile task will visit *e* to perform an intracranial EEG analysis to identify the abnormal area. Finally the abnormal area will be resected and a record will be inserted into a database at host *f* (Figure 2d).

The above example motivates the need for an access control model in which access control policies can be specified based on both the host visit history and future migration behavior of a mobile task.

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. To the best of our knowledge, this is the first time that such a model is proposed.
2. We design a formal itinerary-based access control model. While our previous work [4] 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 are 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.

2. Related work

According to the workflow reference model [5], a workflow describes a business process. Correctness and security are two most important aspects of a workflow management system. An excellent overview of correctness issues in workflow management was given in [6]. Some researchers focus on how to ensure data consistency when concurrency and failures are present. These techniques emerge from the areas of extended transaction models [7-8], multidatabases [9, 10], and transactional workflows [11, 12]. Others focus on the data and control flow requirements. These techniques include control flow graph, triggers (i.e., event-condition-action rules) [13], temporal constraints [14-15], Petri nets [16-17], and set and graph theory [18]. As for security, there is a wide range of security requirements [19]. While process integrity is ensured by constrained planning [20], data confidentiality is often supported by integrating role based access control in the enactment system [21-23]. Security requirements can be either managed by the workflow system itself, or enforced outside of the workflow engine [24].

In recent years, scientific workflows have gained great momentum due to their critical roles in e-Science and cyberinfrastructure applications [25, 26]. There is a plethora of scientific workflows covering a wide range of scientific disciplines. The scale of e-Science computations has motivated researchers to harness distributed environments for running data-intensive scientific applications. Much work has been done towards executing scientific workflows on Grids and Clouds [27]. Thus, Yu and Buyya characterized and classified various approaches for building and executing workflows on the Grid [28] highlighting that scheduling workflow tasks in a distributed Grid environment is a challenging problem. Besides, they proposed their own scheduling approach in [29]. The workflow scheduling problem is showcased in the ASKALON project [30] by Wieczorek et al.

Another aspect of scientific workflow that received considerable attention in the community is workflow and data provenance [31]. To facilitate collaborations and workflow reuse, the problem of workflow annotation and metadata management has been addressed in [32, 33]. Yang et al. propose to use hierarchical state machine to formally model and verify scientific workflow designs [34]. Other research directions in scientific workflows include workflow composition [35], scientific data management inside the resource as well as between different resources [36], workflow security [19, 37], metadata and provenance management [36], including provenance capture, storage querying [38], reusability of scientific workflow system components [39] Nevertheless, none of the above work has addressed the security and correctness issues of scientific workflows that support mobile tasks.

Extensive work has been done on mobile agents [40-44] and their applications [45-51]. There is an increasing interest in the development of itinerary-driven mobile agents [4, 43, 52-54], in which the specification of the mobility behavior of a mobile agent is separated from the specification of its computational behavior.

Many researchers have investigated various security issues in mobile agents [55-56], in particular, the access control at each host [57-59]. 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. This 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 a 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* [4, 54, 60] 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.

In [61] (which extends [4]) we focused on the access control approach that relied on model generation and negotiation between the mobile agent and the host. We used a fragment of the modal μ-calculus [62] to specify access control policies. In this paper, on the other hand, we focused on the itinerary-based access control model that considers host visit history of a mobile task along with its future migration behavior without the option of negotiations. Unlike [61], here, we use Control Tree Logic to specify access control policies and Host Transition Graph to verify the itinerary against the access control policy.

This paper extends [63] with the following additional contributions:

1. We present three new algorithms to verify the access request from a mobile task against the combination of multiple access control policies of the host, which adds flexibility to the proposed model.
2. We introduced an algorithm to create a Host Transition Graph for a given history.
3. We enhanced our access control model with four new satisfaction relationships to verify an itinerary of a mobile task against an access control policy at the host.

3. Access control model

In this section we define the model we use in our workflow system that provides a basis for our access control mechanism.

3.1. Resource model

Our resource model consists of a set of mobile tasks and a set of hosts that these tasks can be executed on. Our model is flexible, i.e. it allows hosts to be geographically spread across multiple locations. It also does not require them to run a particular, or even the same operating system. Our model is agnostic to the programming language used by the mobile tasks. Because our model is not operating system- or language-specific, proposed approach can be applied to a wide range of distributed workflow systems.

As shown on Figure 3, a mobile X consists of computation and itinerary components, where specification of the computational behavior of a mobile task is separated from the specification of its navigational 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.

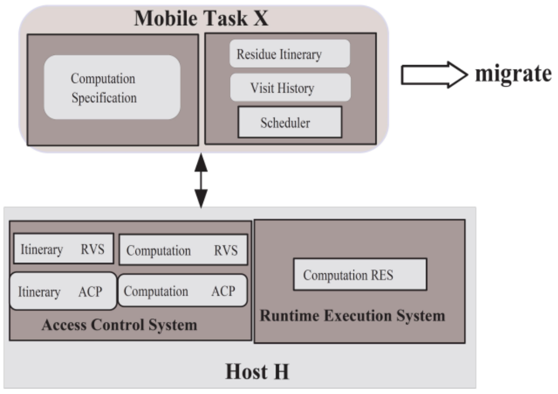


Figure 3: Architecture for mobile task migration and execution.

3.2. Application Model

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 mobile task *X* can execute its computation specification; otherwise, the mobile task *X*'s computation specification will not 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 [64-66] can be applied to handle computation access control.

We now define the problem as follows. Given a mobile task X with its visit history `history` and residue itinerary `residue`, and a host h_0 that X is trying to access, grant X access to h_0 if and only if its host transition graph satisfies access control policy of h_0.

4. Mobile Task Scheduling

In this paper, we consider a simple form of itinerary which has the following BNF syntax.

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

where *s* represents a single host visit and ;, #, and || are 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 to as *P.h*, is a host that a mobile task may visit next, and i_k, referred

to 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 (_, Ø) where _ is 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 Ø, is the list of hosts that the mobile task has 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 $M$;
```

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 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. 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 backtracking in the future. This is because the access of host next does not guarantee 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.

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
```

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 efficient 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 specifying itinerary-based Access Control Policies (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 two path quantifiers: A (for all paths) and E (for some path); and four temporal operators: X(next time), F(eventually), G(always), U(until). Formally, the formulas that describe the future behaviors of ACP are defined by the following grammar.

$$\mu ::= h \mid \mathsf{EX}\mu \mid \mu\mathsf{EU}\mu \tag{1}$$

where h is a host.

Note that the unary temporal connective EX (*possibly-next*), and the binary temporal connective EU (*possibly-until*) can be used to define other connectives:

- *possibly-eventually*: $\mathsf{EF}\mu$ for $true\mathsf{EU}\mu$;
- *Inevitably-next*: $\mathsf{AX}\mu$ for $\neg\mathsf{EX}\neg\mu$;
- *Inevitably-always*: $\mathsf{AF}\mu$ for $\neg\mathsf{EF}\neg\mu$

In order to reason about past-time behaviors, we introduce the following past temporal operators: Y (in the previous time instance), P (sometime in the past), H (always in the past), 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. The past-time behaviors is defined by the following grammar.

Definition 2 [**Access Control Policy – Past Behaviors**]

$$v ::= h \mid \mathsf{AY}v \mid v\mathsf{AS}v \tag{2}$$

Similarly, the unary temporal connective AY (*previous*), and the binary temporal connective AS (*since*) can be used to define other connectives:

- *past*: $\mathsf{AP}v$ for $true\mathsf{AS}v$;
- *Inevitably-past*: $\mathsf{AH}v$ for $\neg\mathsf{AP}\neg v$

Finally, the access control policy is defined with the following syntax.

Definition 3 [Access Control Policy]

$$\phi ::= \mu \mid v \mid \phi \vee \phi \mid \neg \phi \tag{3}$$

Example 5.1 The access control policies described informally in Example 1.1 can be formally specified using ACP formulas as follows.

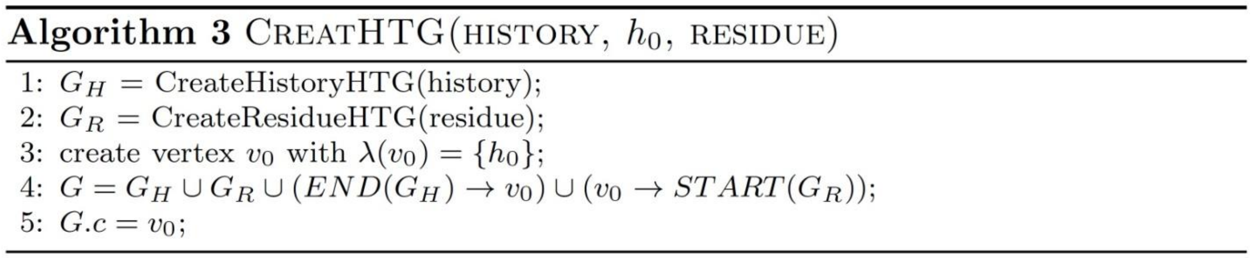
1. $\mathsf{AP}e$: a mobile task can visit current host only if it has visited *e*.
2. $\mathsf{AP}e \wedge \mathsf{AP}\,f$: a mobile task can visit current host only if it has visited both *e* and *f* in any order.

3. AP(f ∧ APe): 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).
4. AP(b ∧ AYa) ∧ EF($e \rightarrow$ AFf): a mobile task can visit current host h only if it has visited a and then b in the past without visiting any other hosts in between, and in the future if the mobile task visits e then it will also visit f later eventually.

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 = $<V, T, \lambda, c>$ 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-visited-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 shows the pseudo-code on how to construct a HTG based on three inputs: `history` are the hosts that have 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 links the two components through the to-be-visited-next vertex v_0 that is labeled with the set $\{h_0\}$.


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: $G.c = v_0$;
```

The algorithm to create HTG for history is presented in Algorithm 4. A linear graph is generated with each vertex labeled by a singleton set. Algorithm 5 shows how to construct the HTG component for `residue`. It is straight forward 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 $START(G_2) = \{s_3, s_4\}$, the following four edges are added: $s_1 \rightarrow s_3$, $s_2 \rightarrow s_3$, $s_1 \rightarrow s_4$, $s_2 \rightarrow s_4$. 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

Algorithm 4 HTG CreateHistoryHTG(history)

```
1: Let history=$\{h_n, \ldots, h_1\}$;
2: for all $h_i \in$ history do
3:     add a vertex $v_i$ with $\lambda(v_i) = \{h_i\}$;
4: end for
5: $G_H.T = \{(h_n \rightarrow h_{n-1}), \ldots, (h_2 \rightarrow h_1)\}$;
6: $END(G_H) = v_1$;
7: return $G_H$;
```

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.

Example 5.2 Assume that a mobile task who tries to access host h_3 has visited hosts h_1, h_2 in that order, and the residue itinerary is ($h_4||h_5$); ($h_6\#h_7$). The host transition graph is shown in Figure 4, where the visit history consists of nodes marked with set h_1 or h_2, the to-be-visited-next node is marked with set h_3, and the residue consists of nodes marked with sets h_4 or h_5. The semantics of access control policy can be explained on host transition graphs. When a mobile task X requests for access at host h_0, the itinerary tuple <*history*, h_0, *residue*> of X will be used to construct an HTG G_X. Note that initially each vertex in G_X is labeled with a singleton set and $G_X.c$ is labeled with $\{h_0\}$. It is also possible that multiple vertices in G_X are labeled with the same host name.

Algorithm 5 HTG CREATRESIDUEHTG(RESIDUE)

```
if Residue == $s$ then
    add a vertex $v$ to $G$ with $\lambda(v) = \{h\}$;
    $START(G) = END(G) = \{v\}$;
else if Residue == $i_1; i_2$ then
    $G_1$ = CreatResidueHTG($i_1$);
    $G_2$ = CreatResidueHTG($i_2$);
    $G = G_1 \cup G_2 \cup (END(G_1) \rightarrow START(G_2))$;
    $START(G) = START(G_1)$;
    $END(G) = END(G_2)$;
else if Residue == $i_1 \# i_2$ then
    $G_1$ = CreatResidueHTG($i_1$);
    $G_2$ = CreatResidueHTG($i_2$);
    $G = G_1 \cup G_2$;
    $START(G) = START(G_1) \cup START(G_2)$;
    $END(G) = END(G_1) \cup END(G_2)$;
else if Residue == $i_1 || i_2$ then
    $G_1$ = CreatResidueHTG($i_1$);
    $G_2$ = CreatResidueHTG($i_2$);
    for all $s_1 \rightarrow s_2 \in G_1$ do
        for all $t_1 \rightarrow t_2 \in G_2$ do
            $G = G \cup \{s_1 \rightarrow s_2 \times t_1 \rightarrow t_2\}$;
        end for
    end for
    $START(G)$ = GetStartSet($G$);
    $END(G)$ = GetEndSet($G$);
end if
return G;
```

The set of access control policy formulas labeled at each vertex will be changed in the algorithms introduced in Section 5.3.

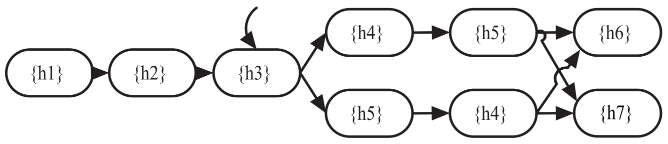


Figure 4: A host transition graph.

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

Definition 4 [**Satisfaction relationship** |=] Let v_0 be a vertex in G_X created from the itinerary of mobile task 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 $v_0 \models \phi_2$.
- $v_0 \models \mathsf{AX}\phi$ iff for all vertices v such that $(v_0 \rightarrow v) \in G$, $v \models \phi$.
- $v_0 \models \mathsf{EX}\phi$ iff for some vertices v such that $(v_0 \rightarrow h) \in G$, $v \models \phi$.
- $v_0 \models \mathsf{AF}\phi$ iff for all paths $(v_0, v_1, \ldots)$, $\exists i[i \geq 0 \wedge v_i \models p]$.
- $v_0 \models \mathsf{EF}\phi$ iff for some paths $(v_0, v_1, \ldots)$, $\exists i[i \geq 0 \wedge v_i \models p]$.
- $v_0 \models \phi_1 \mathsf{AU} \phi_2$ iff for all paths $(v_0, v_1, \ldots)$, $\exists i[i \geq 0 \wedge v_i \models \phi_2 \wedge \forall j[0 \leq j < i \rightarrow v_i \models \phi_1]]$
- $v_0 \models \phi_1 \mathsf{EU} \phi_2$ iff for some paths $(v_0, v_1, \ldots)$, $\exists i[i \geq 0 \wedge v_i \models \phi_2 \wedge \forall j[0 \leq j < i \rightarrow v_i \models \phi_1]]$
- $v_0 \models \mathsf{AY}\phi$ iff for the vertex v such that $(v, v_0) \in \mathrm{R}$, $v \models \phi$.
- $v_0 \models \mathsf{AP}\phi$ iff for the path $(\ldots, v_1, v_0)$, $\exists i[i \geq 0] \wedge v_i \models \phi$.
- $v_0 \models \phi_1 \mathsf{AS} \phi_2$ iff for the path $(\ldots, v_1, v_0)$, $\exists i[i \geq 0 \wedge vi \models \phi_2 \wedge \forall j[0 \leq j < i \rightarrow \mathrm{v}_i \models \phi_1]]$

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

$$Sub(h) = \{h\} \text{if } h \text{ is a host}$$

$$Sub(\phi_1 \vee \phi_2) = \{ \phi_1 \vee \phi_2\} \cup Sub(\phi_1) \cup Sub(\phi_2)$$

$$Sub(\neg\phi) = \{\neg\phi\} \cup Sub(\phi)$$

$$Sub(\mathsf{EX}\phi) = \{\mathsf{EX}\phi\} \cup Sub(\phi)$$

$$\mathrm{Sub}(\phi_1\mathsf{EU}\phi_2) = \{\phi_1\mathsf{EU}\phi_2\} \cup Sub(\phi_1) \cup Sub(\phi_2)$$

$$Sub(\mathsf{AY}\phi) = \{\mathsf{AY}\phi\} \cup Sub(\phi)$$

$$Sub(\phi_1\mathsf{AS}\phi_2) = \{\phi_1\mathsf{AS}\phi_2\} \cup Sub(\phi_1) \cup Sub(\phi_2)$$

$OrderedSub(\phi)$ 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 6 [**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 $v \in [\phi]_G \leftrightarrow \phi \in \lambda(v)$.

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 6. The for loop (Lines 1-15) iterates over all the subformulas of ϕ and calls functions (Algorithms 7-10) 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*(ϕ).

Algorithm 6 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);
```

Finally (Line 16), 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 the h_0 will reject the request. This is because $G.c \models \phi$ iff $h_0 \in [\phi]_G$. The algorithms for *AccessControlAY* and *AccessControlAS* are omitted since they are relative simple, due to the fact that *G* is a linear structure.

Algorithm 7 shows the function to check a formula with format $\phi_1 \vee \phi_2$. Since ϕ_1 and ϕ_2 are subformulas of $\phi_1 \vee \phi_2$, they must have been checked before. The algorithm iterates all the vertices $v \in G$. If either ϕ_1 or ϕ_2 is a member of $\lambda(v)$, $\phi_1 \lor \phi_2$ becomes a member as well. Similarly, Algorithm 8 shows how to check formulas with Boolean connective $\neg$.

Algorithm 9 shows the function to check formulas with format $\mathsf{EX}\phi_1$. For each $v \in G$, if one of its successors v' has $\phi_1 \in \lambda(v')$, then $\mathsf{EX}\phi_1 \in \lambda(v)$ due to the semantics of EX.

Algorithm 10 shows how to check formulas of the form $\phi_1\mathsf{EU}\phi_2$. The basic idea is to first find out all the hosts v such that $\phi_2 \in \lambda(v)$. Then starting from these hosts the algorithm does a backward traversal on all the hosts v' such that $\phi_1 \in \lambda(v')$. For such hosts, we have $\phi_1\mathsf{EU}\phi_2 \in \lambda(v')$. The traversal stops on a path whenever $\phi_1 \notin \lambda(v)$. In order to do so we need help of a set α and a stack β that are initially empty (line 1).

Algorithm 7 ACCESSCONTROLOR(HTG G, POLICY $\phi_1 \lor \phi_2$)

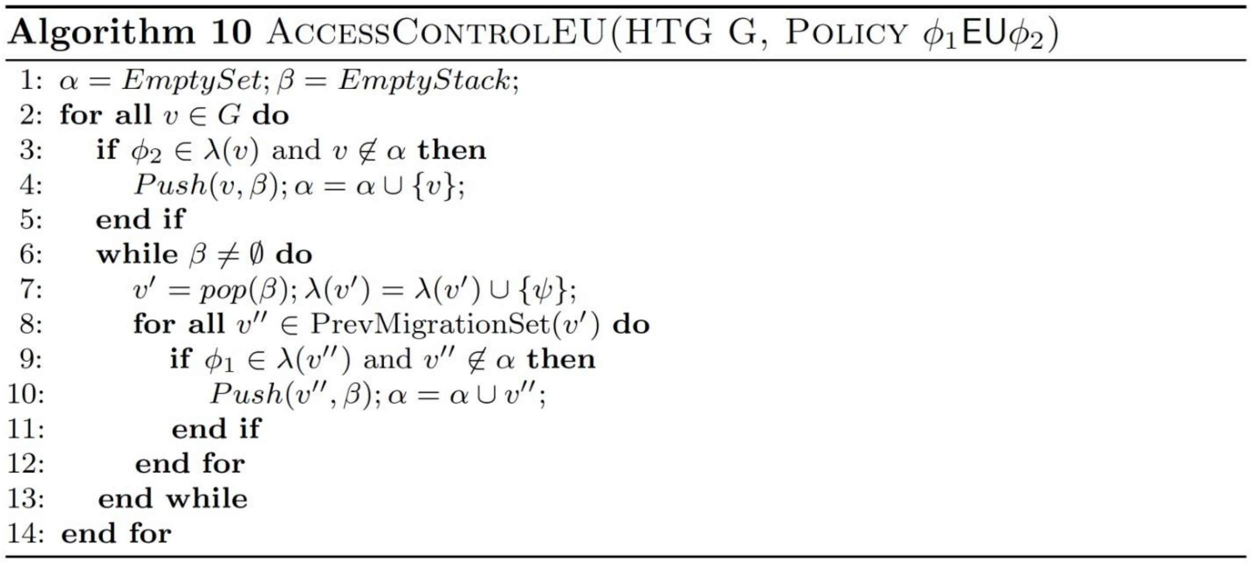
```
1: for all $v \in G$ do
2:     if $\phi_1 \in \lambda(v)$ or $\phi_2 \in \lambda(v)$ then
3:         $\lambda(v) = \lambda(v) \cup (\phi_1 \lor \phi_2)$;
4:     end if
5: end for
```

Algorithm 8 ACCESSCONTROLNOT(HTG G, POLICY $\neg\phi_1$)

```
1: for all $v \in G$ do
2:     if $\phi_1 \notin \lambda(v)$ then
3:         $\lambda(v) = \lambda(v) \cup (\neg\phi_1)$;
4:     end if
5: end for
```

Algorithm 9 ACCESSCONTROLEX(HTG G, POLICY $\mathsf{EX}\phi_1$)

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



Algorithm 10 ACCESSCONTROLEU(HTG G, POLICY $\phi_1 \mathsf{EU} \phi_2$)

```
1:  $\alpha = EmptySet; \beta = EmptyStack;$
2:  for all $v \in G$ do
3:      if $\phi_2 \in \lambda(v)$ and $v \notin \alpha$ then
4:          $Push(v, \beta); \alpha = \alpha \cup \{v\};$
5:      end if
6:      while $\beta \neq \emptyset$ do
7:          $v' = pop(\beta); \lambda(v') = \lambda(v') \cup \{\psi\};$
8:          for all $v'' \in \mathrm{PrevMigrationSet}(v')$ do
9:              if $\phi_1 \in \lambda(v'')$ and $v'' \notin \alpha$ then
10:                 $Push(v'', \beta); \alpha = \alpha \cup v'';$
11:             end if
12:         end for
13:     end while
14: end for
```

For each $v \in G$, if $\phi_2 \in v$ and v has not been visited before, it is added to both α and β (lines 3-5). The while loop (lines 6-13) repeats until the stack becomes empty. It first pops β to get v' and append $\lambda(v')$ with $\phi_1 \mathsf{EU} \phi_2$. Then continues backward traversal from v'. Note function *PrevMigrationSet*(v') returns all the hosts v'' such that $v'' \rightarrow v'$ is an edge of G.

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 6 solves the access control problem in $O(\rho \times (n + m))$ time, and requires $O(\rho \times n)$ space.*

Example 5.3 We now apply a verification policy from Example 1.1 to a specific host transition graph. Consider a mobile task A whose host transition graph G_A is shown in Figure 5. A is requesting to access host h with the policy

$$\phi = \mathsf{AP}e$$

which is a formal specification of the policy in the first application from Example 1.1 that states that a mobile task can visit current host h only after visiting host *e* (DTI analysis application). Host h will grant X access if the following relationship is satisfied:

$h \models \mathrm{AP}e$

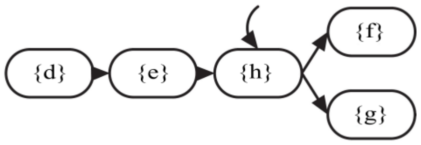


Figure 5. Host transition graph G_A

According to Definition 4, such policy is satisfied if and only if there exists a path $(\ldots, v_1, v_0)$, such that for some i $v_i \models e$. From Figure 5, we know that there exists a path (d, e, h), and if i=1, then $v_1 = e$, and $e \models e$. Therefore, the above policy is satisfied by G_A and access is granted to X.

The other three policies in Example 1.1 can be specified as $\mathrm{AP}e \wedge \mathrm{AP}f$, $\mathrm{AP}\,(e \wedge \mathrm{AP}f)$, $\mathrm{AP}\,(a \wedge \mathrm{AP}b) \wedge \mathrm{AF}(e \wedge \mathrm{AF}f)$, respectively. Host h will not grant X access for the following reasons:

- $\mathrm{AP}e \wedge \mathrm{AP}f$ and $\mathrm{AP}\,(e \wedge \mathrm{AP}f)$: no host *f* has been visited in the past.
- $\mathrm{AP}\,(a \wedge \mathrm{AP}b) \wedge \mathrm{AF}(e \wedge \mathrm{AF}f)$: neither host *a* or *b* has been visited in the past, and there is no host *e* to be visited in the future.

6 Conclusions and Future Work

We have presented an itinerary-based access control model to address the need for secure data-intensive distributed scientific workflows. To support secure migration, our proposed model not only considers the host visit history of a mobile task but also its future migration behavior that is prescribed by the residue itinerary. In addition, we proposed a mobile task scheduling algorithm with the salient feature of supporting backtracking to dynamically schedule a successful route of hosts if it exists.

Several directions can be pursued in the future work. First, itinerary language can be extended with additional constructs such as cloning and loop. Second, we will extend our access control model to control fine-grained access to scientific datasets, such as scientific XML datasets and data streams, where the issue is to control who can access which datasets or data elements. Third, we will investigate the access control issues in collaborative scientific workflows, in which a consortium can be formed by several member institutions for a collaborative scientific study. In this context, a collaborative scientific workflow might integrate various services, applications, tools, and datasets that are contributed by different member institutions of the consortium. A consortium based access control system is needed in order to enforce the secure sharing policies that are agreed by all the parties. Such an access control system will be used to control which user, which institution, and which workflow, can access which service, which dataset, and/or which data elements. Finally, we will develop system architecture to support mobile tasks in data-intensive scientific workflows, and will implement our access control model to support secure migration of these tasks.

References

1. Ludascher, B. and C. Goble, Guest editor’s introduction to the special section on scientific workflows. SIGMOD Record, 2005. 34(3): p. 3–4.
2. Lu, S., Semantic Correctness of Transactions and Workflows, 2002, State University of New York at Stony Brook.
3. Lu, S., A. Bernstein, and P. Lewis, Completeness and realizability: Conditions for automatic generation of workflows. International Journal of Foundations of Computer Science, 2006. 17(1): p. 223-245.
4. Yang, Z., S. Lu, and P. Yang. Runtime security verification for itinerary-driven mobile agents. in 2nd IEEE International Symposium on Dependable, Autonomic and Secure Computing. 2006. Indianapolis, USA. p. 177-186.
5. Hollingsworth, D., The workflow reference model, 1995, Workflow Management Coalition.
6. Kamath, M. and K. Ramamritham., Correctness issues in workflow management. Distributed Systems Engineering Journal, 1996. 3(4): p. 213–221.
7. Elmagarmid, A., Database Transaction Models For Advanced Applications, 1992, Morgan Kaufmann Publishers.
8. Wachter, H. and A. Reuter, The Contract Model, in Elmagarmid (ed.): Advanced Transaction Models for New Applications, 1992. p. 220–263.
9. Mehrotra, S., et al. The concurrency control problem in multidatabases: Characteristics and solutions. in ACM SIGMOD Conference on the Management of Data. 1992. San Diego, CA. p. 288-297.

10. Breitbart, Y., H. Garcia-Molina, and A. Silberschatz, Overview of multidatabase transaction management. Journal of VLDBJ, 1992. 1(2): p. 181–240.
11. Sheth, A. and M. Rusinkiewicz, On transactional workflows. Bulletin of the Technical Committee on Data Engineering, 1993. 16: p. 37-40.
12. Zimmermann, O., et al., Architectural decisions and patterns for transactional workflows in SOA, in ICSOC, 2007. p. 81-93.
13. Dayal, U., M. Hsu, and R. Ladin. Organizing Long-running Activities with Triggers and Transactions. in ACM SIGMOD Conference on the Management of Data. 1990. Atlantic City, New Jersey, USA. p. 204–214.
14. Singh, M.P. Semantical considerations on workflows: An algebra for intertask dependencies. in International Workshop on Data Programming Languages. 1995. Gubbio, Umbria, Italy. p. 5.
15. Singh, M.P. Synthesizing distributed constrained events from transactional workflow specifications. in 12th International Conference on Data Engineering. 1996. p. 616–623.
16. Salimifard, K. and M. Wright, Petri net-based modelling of workflow systems: An overview. European Journal of Operational Research, 2001. 134(3): p. 664-676.
17. Aalst, W.M.P.v.d., A. Hirnschall, and H.M.W.E. Verbeek, An alternative way to analyze workflow graphs. CAiSE, 2002: p. 535–552.
18. Arpinar, I.B., et al., Formalization of Workflows and Correctness Issues in the Presence of Concurrency. Distributed and Parallel Databases, 1999. 7(2): p. 199–248.
19. Atluri, V. and J. Warner, Security for workflow systems, in Handbook of Database Security, 2008. p. 213–230.

20. Wainer, J., P. Barthelmess, and A. Kumar, W-RBAC - A Workflow Security Model Incorporating Controlled Overriding of Constraints. International Journal of Cooperative Information Systems, 2003. 12(4): p. 455–485.
21. Kang, M., J. Park, and J. Froscher. Access control mechanisms for inter-organizational workflow. in Sixth ACM Workshop on Role Based Access Control. 2001. Chantilly, Virginia. p. 66–74.
22. Park, S.-H., J.-H. Eom, and T.-M. Chung, A study on access control model for context-aware workflow. NCM, 2009: p. 1526-1531.
23. Hung, P. and K. Karlapalem. A secure workflow model. in Australasian information security workshop conference on ACSW frontiers. 2003. Adelaide, Australia. p. 33-41.
24. Chivers, H. and J. McDermid, Refactoring service-based systems: How to avoid trusting a workflow service. Concurrency and Computation, 2006. p. 1255-1275.
25. Deelman, E., et al., Workflows and e-science: An overview of workflow system features and capabilities. Future Generation Comp. Syst., 2009. 25(2): p. 528-540.
26. Ludascher, B. and C. Goble, Guest editor’s introduction to the special section on scientific workflows. SIGMOD Record, 2005. 34(3): p. 3-4.
27. Deelman, E., Grids and clouds: Making workflow applications work in heterogeneous distributed environments. IJHPCA, 2010. 24(3): p. 284-298.
28. Yu, J. and R. Buyya, A taxonomy of scientific workflow systems for grid computing. SIGMOD Record, 2005. 34(3): p. 44-49.
29. Yu, J., M. Kirley, and R. Buyya, Multi-objective planning for workflow execution on grids. GRID, 2007: p. 10-17.

30. Wieczorek, M., R. Prodan, and T. Fahringer, Scheduling of scientific workflows in the ASKALON grid environment. SIGMOD Record, 2005. 34(3): p. 57-62.

31. Davidson, S.B. and J. Freire, Provenance and scientific workflows: challenges and opportunities, in SIGMOD Conference, 2008. p. 1345–1350.

32. Gandara, A., et al., Knowledge annotations in scientific workflows: An implementation in Kepler. SSDBM, 2011: p. 189-206.

33. Medeiros, C.B., et al., WOODSS and the web: Annotating and reusing scientific workflows. SIGMOD Record, 2005. 34(3): p. 18-23.

34. Yang, P., Z. Yang, and S. Lu. Formal modeling and analysis of scientific workflows using hierarchical state machines. in Second IEEE International Workshop on Scientific Workflows and Business Workflow Standards in e-Science, in conjunction with e-Science'07. 2007. p. 619-626.

35. Fei, X., S. Lu, and C. Lin, A MapReduce-enabled scientific workflow composition framework, in ICWS, 2009. p. 663-670.

36. Deelman, E. and A.L. Chervenak, Data management challenges of data-intensive scientific workflows. CCGRID, 2008: p. 687-692.

37. Hsiao, Y.-C. and G.-H. Hwang, Implementing the chinese wall security model in workflow management systems, in ISPA, 2010. p. 574-581.

38. Lim, C., et al., Storing, reasoning, and querying OPM-compliant scientific workflow provenance using relational databases. Future Generation Comp. Syst., 2011. 27(6): p. 781-789.

39. Lin, C., et al., A reference architecture for scientific workflow management systems and the view SOA solution. IEEE T. Services Computing, 2009. 2(1): p. 79-92.

40. Gray, R., et al., Mobile agents: Motivations and state-of-the-art systems. Handbook of Agent Technology, 2000.

41. Xu, C.-Z. Naplet: A flexible mobile agent framework for network-centric applications. in 2nd Int'l Workshop on Internet Computing and E-Commerce (In conjunction with IEEE IPDPS'2002). 2002.

42. Tsai, H.-W., C.-P. Chu, and T.-S. Chen, Mobile object tracking in wireless sensor networks. Computer Communications, 2007. 30(8): p. 1811-1825.

43. Tripathi, A., N. Karnik, and T. Ahmed, Design of the Ajanta system for mobile agent programming. Journal of Systems and Software, 2002. p. 123-140.

44. Gavalas, D., G.E. Tsekouras, and C. Anagnostopoulos, A mobile agent platform for distributed network and systems management. Journal of Systems and Software, 2009. 82(2): p. 355-371.

45. Chen, B. and W. Liu, Mobile agent computing paradigm for building a flexible structural health monitoring sensor network. Comp.-Aided Civil and Infrastruct. Engineering, 2010. 25(7): p. 504-516.

46. Fok, C.-L., G.-C. Roman, and C. Lu, Agilla: A mobile agent middleware for self-adaptive wireless sensor networks. TAAS, 2009. 4(3).

47. Xu, C. and B. Wims, Mobile agent based push methodology for global parallel computing. Concurrency: Practice and Experience, 2000. 14(8): p. 705-726.

48. Kona, M. and C. Xu. A framework for network management using mobile agent approach. in First IEEE Int'l Workshop on Internet Computing and E-Commerce. 2001. San Francisco, CA, USA.

49. Chen, B., H.H. Cheng, and J. Palen, Integrating mobile agent technology with multi-agent systems for distributed traffic detection and management systems. Transportation Research Part C: Emerging Technologies, 2009. 17(1): p. 1-10.

50. Blake, M., Agent-oriented approaches to B2B interoperability. The Knowledge Engineering Review, 2001. 16(4). p. 1-5.

51. Sandholm, T., Agents in electronic commerce: Component technologies for automated negotiation and coalition formation. Autonomous Agents and Multi-Agent Systems, 2000. 3(1): p. 73-96.

52. Chen, M., et al. Energy-efficient itinerary planning for mobile agents in wireless sensor networks. in IEEE International Conference on Communications. 2009. Dresden, Germany. p. 1-5.

53. Xu, Y. and H. Qi, Mobile agent migration modeling and design for target tracking in wireless sensor networks. Ad Hoc Networks, 2008. 6(1): p. 1-16.

54. Lu, S. and C.Z. Xu, A formal framework for agent itinerary specification, security reasoning and logic analysis, in ICDCS Workshops, 2005. p. 580-586.

55. Burkle, A., et al., Evaluating the security of mobile agent platforms. Autonomous Agents and Multi-Agent Systems, 2009. 18: p. 295-311.

56. Ma, L. and J.J.P. Tsai, Formal modeling and analysis of a secure mobile-agent system. IEEE Transactions on Systems, 2008. 38(1): p. 180-196.

57. Cao, C. and J. Lu. A path-history-sensitive access control model for mobile agent environment. in Third International Workshop on Mobile Distributed Computing (MDC) (ICDCSW’05). 2005. Washington, DC: IEEE Computer Society. p. 660-663.

58. Chung, Y.-F., T.-S. Chen, and M.-W. Lai, Efficient migration access control for mobile agents. Computer Standards & Interfaces, 2009. 31(6): p. 1061-1068.

59. Navarro, G., et al. Access control with safe role assignment for mobile agents. in Fourth International Joint Conference on Autonomous Agents and Multiagent Systems. 2005. New York, NY, USA: ACM Press. p. 1235-1236.

60. Lu, S., Itinerary safety reasoning and assurance. Scalable and Secure Internet Services and Architecture, 2005: p. 247-262.

61. Yang, Z., et al., Model Checking Approach to Secure Host Access Enforcement of Mobile Tasks in Scientific Workflows. International Journal of Computers and Their Applications, 2011. 18(3): p. 148-159.

62. Kozen, D., Results on the Propositional mu-Calculus. Theoretical Computer Science, 1983. 27: p. 333-354.

63. Yang, Z., S. Lu, and P. Yang. Itinerary-Based Access Control for Mobile Tasks in Scientific Workflows. in International Conference on Advanced Information Networking and Applications Workshops. 2007. Ontario, Canada. p. 506-511.

64. Ivancic, F., et al., F-Soft: Software verification platform. Computer-Aided Verification, 2005: p. 301-306.

65. C.Wang, et al., Disjunctive image computation for emebedded software verification, in Design, Automation and Test in Europe (DATE’06)2006: Munich, Germany. p. 1205-1210.

66. Yang, Z., et al., Mixed symbolic representations for model checking software programs, in ACM/IEEE International Conference on Formal Methods and Models for Codesign (Memocode’06), 2006. p. 17-26.